FS19

BLOCKCHAIN, BITCOIN, ETHEREUM III

Ethereum Smart Contracts II

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

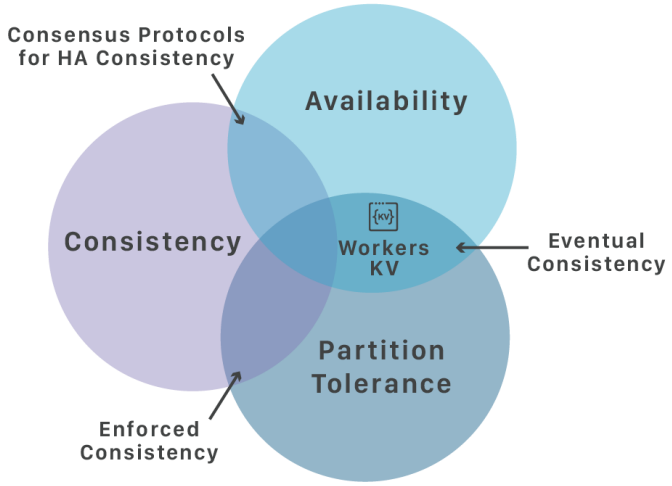
FHO Fachhochschule Ostschweiz

Agenda

- **Distributed Systems - In the News**
- **Solidity**
  - Examples
  - ERC20
  - Security Considerations
  - Ganache/Testonator

Distributed Systems - In the News

- **21.05.2019: Workers KV — Cloudflare's distributed database**
  - Ready for production use
  - Distributed, eventually consistent, key-value store
  - Read values with ultra-low latency ~12ms anywhere in the world
  - Cloudflare's global edge of 175+ data centers
  - CAP theorem: AP
  - Write ~60sec



  - If two clients write different values to the same key at the same time, the last client to write eventually "wins“
  - Key – value – up to 2 MB, can also be JavaScript

- **May 2019: Finding Something We Can All Agree On: An Overview of Distributed Systems and the Consensus Problem**
  - Coordination Protocols and Paradigms
    - Eventual Consistency Protocols
      - Read Repair, Asynchronous Repair (Cassandra), No Coordination (CRDT)
    - Strong Consistency Protocols
      - Two-Phase Commit, Paxos
  - Applications of Consensus
    - Mutual Exclusion, Leader Election, Atomic Broadcast

Learning Goals

- **I saw the syntax of Solidity and know where to find more information**
- **I can deploy a Solidity smart contract**
- **I can write a simple Solidity smart contract**
- **I know the common pitfalls**

Solidity Contracts

- **Notary Contract**
  - Simple Contract
  - No constructor
- **Frontend**
  - Vue.js, dependencies:
    - crypto-js
    - vue
    - web3
- **App.vue, main file**
  - Template
  - Create web3 instance
  - Events
    - fileChange()
    - store()

```
pragma solidity ^0.5.7;

contract Notary {

    mapping (address => mapping (bytes32 => uint256)) stamps;

    function store(bytes32 hash) public {
        stamps[msg.sender][hash] = block.timestamp;
    }

    function verify(address recipient, bytes32 hash)
                            public view returns (uint256) {
        return stamps[recipient][hash];
    }
}
```

Solidity: Version, Mapping

```
/* Specifies what versions of Solidity a
 * source file can work on
 */
pragma solidity ^0.5.7;


contract Notary {

    …
}
```

- **Ensures that the source code is only meant for compiler versions that are not older than 0.5.0 and will also not work on a compiler starting from version 0.6.0.**
- **Mapping of address, of mapping of hash, and timestamp**

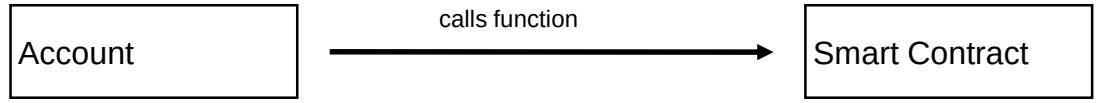
Address	Hash	Timestamp
0xD6df5935cD03a768b7B9E92637A01b25e24cb709	ed52f3833d5d2c920fd43a0972add3555ad149f0201bbd134907498a851a5ec3	2009-06-24 12:39:54 +0900
0xD6df5935cD03a768b7B9E92637A01b25e24cb709	0c911c1d3bde4be991e3d39c2f7e97c621617feaf8f7070e2384c602430ee382	2015-04-14 10:34:24 +0800
0x1530df3e1C69501d4Ecb7E58eB045b90DE158873	0c911c1d3bde4be991e3d39c2f7e97c621617feaf8f7070e2384c602430ee382	2018-07-21 14:33:24 +0900
…	…	…

```
contract Notary {
  mapping (address => mapping (bytes32 => uint256)) stamps;
  …
}
```

Solidity: Constructor / Function Calls

```
contract  Notary {
    address owner;
    /* Initializes the contract, set owner */
    constructor (uint256 number) public {
        owner = msg.sender;
        balances[msg.sender] = number;
     }
     …
}
contract MyToken {
     …
      function store(bytes32 hash) public {
         …
     }
}
```

- **The constructor is run only once, i.e., when the contract is created!**
- **Smart contract function can call another smart contract function**
- **Initiator always needs to be an external account**
  - Externally owned account vs. contract account



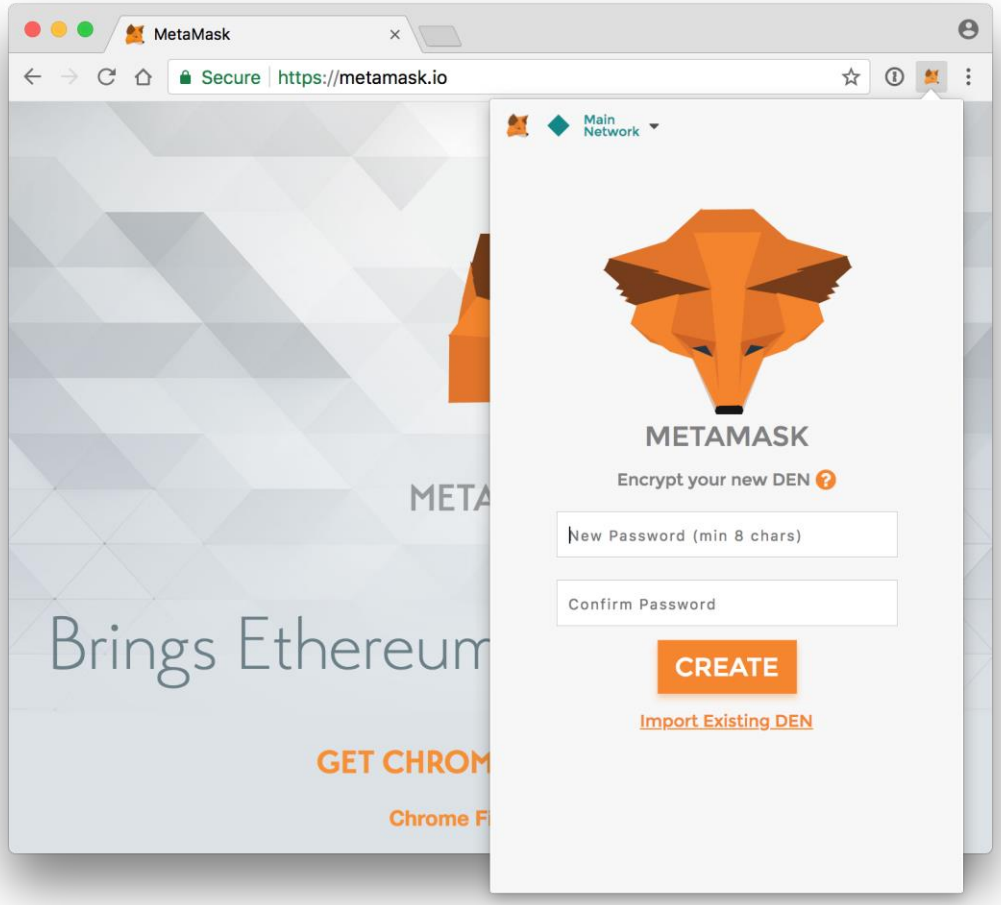
OR

URL: b.socrative.com/student/

Room: HSR

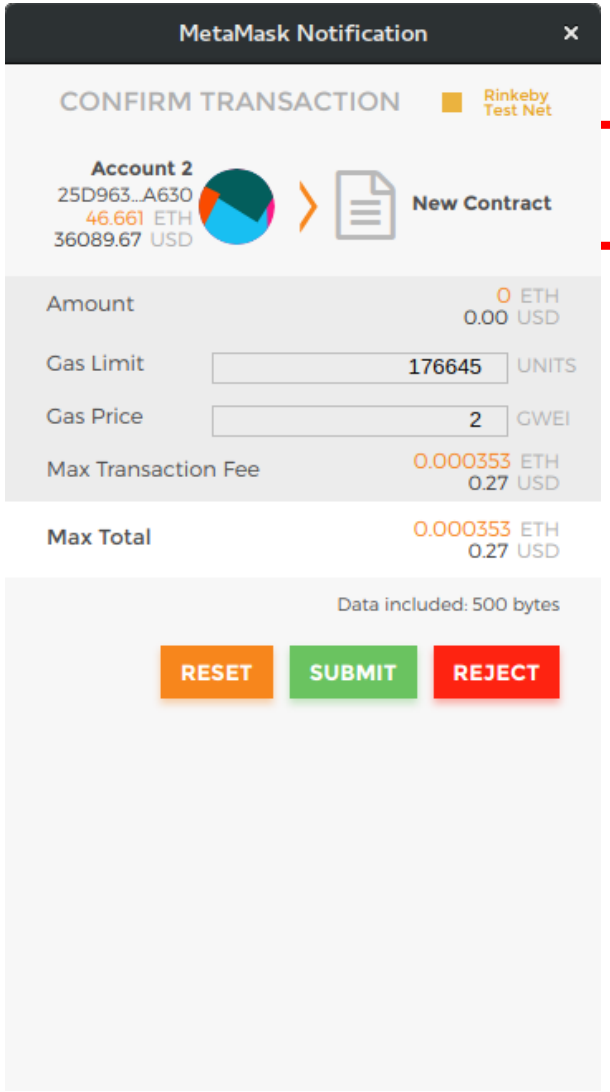
MetaMask

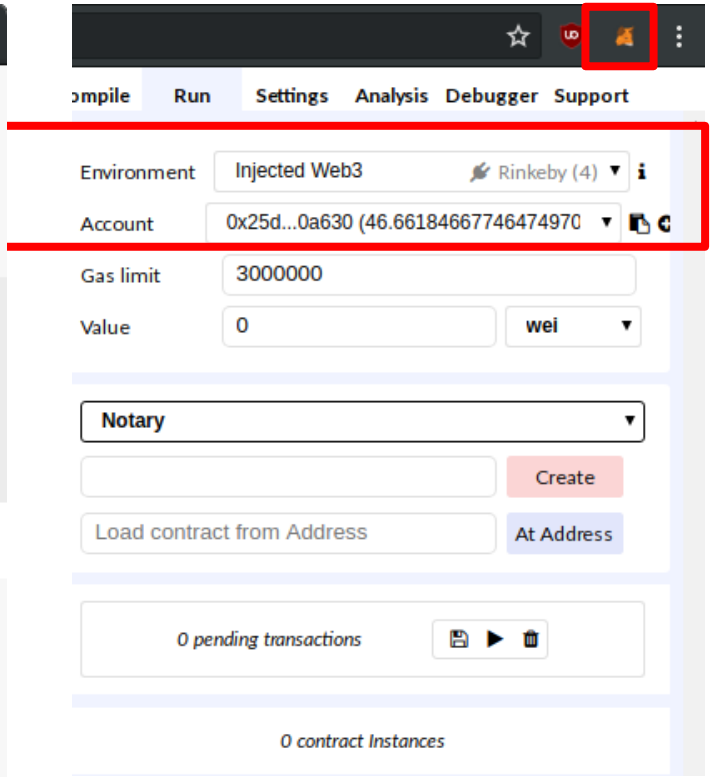
- **MetaMask**
  - Browser plugin to make Ethereum transactions in browsers
  - Manage your key pairs and sign blockchain transactions (is a wallet, BIP39)
  - MetaMask injects javascript library - web3.js
  - Uses infura
- **Open Remix IDE: https://remix.ethereum.org**
  - Use Notary.sol from https://github.com/tbocek/VSS-WEB3/blob/master/Notary.sol
  - Alternatively, use IntelliJ solidity plugin and deploy via geth, parity, or a local test blockchain
- **Compile:**
  - browser/Notary.sol:8:36:use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.



Create Contract

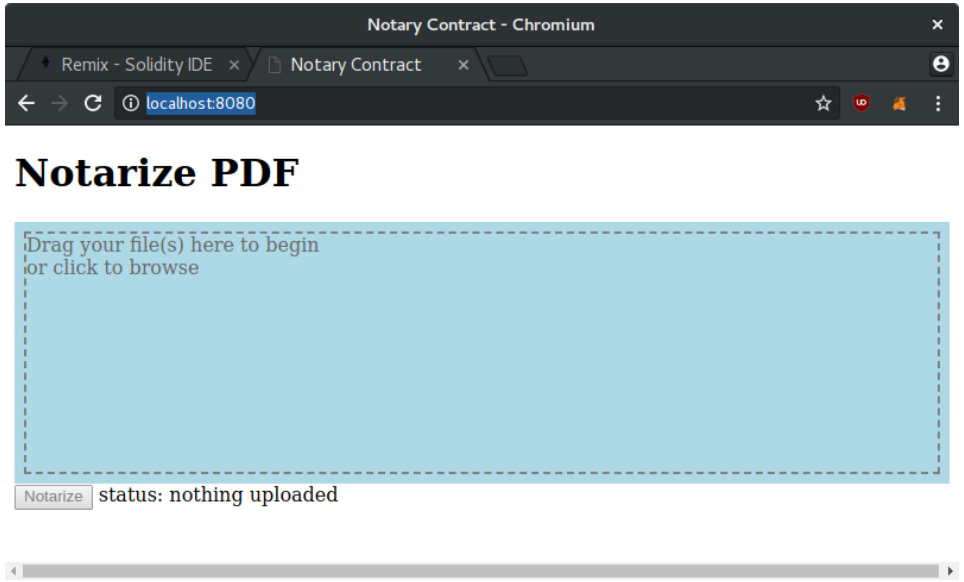
- **Connect to MetaMask**
  - Injected Web3
  - If you see your accounts, good!
- **Create contract!**
  - Add address to the code (manually)
- **Store value**
  - 0x654cf88c**4cff3e62696f7cbb55fbba922b2a75897a010347e371e9398b658c4affa611aa**
  - Hash is: 0x4cff3e62696f7cbb55fbba922b2a75897a010347e371e9398b658c4affa611aa
  - What is 0x654cf88c?
- **First four bytes of the Keccak (SHA-3) hash of the signature of the function.**
  - Keccak(store(bytes32))

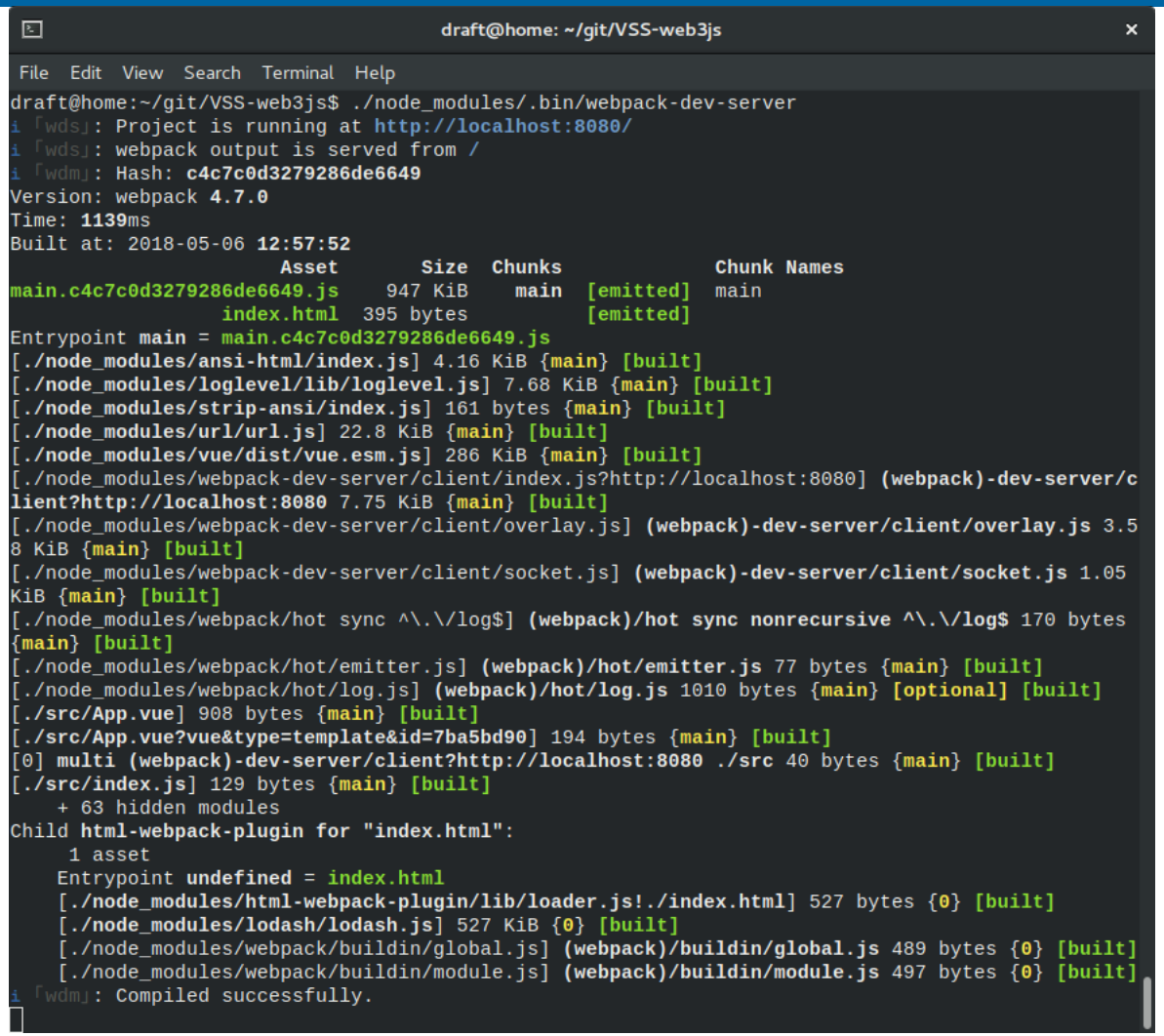




Run it locally

- **Installation**
  - yarn install
  - ./node_modules/.bin/webpack-dev-server
- **Open Browser: http://localhost:8080/**





URL: b.socrative.com/student/

Room: HSR

ERC20

- **ERC20 is a technical standard for smart contracts on the Ethereum blockchain for implementing tokens**
  - proposed on November 19, 2015
  - January 2018: more than 21000 ERC20 token contracts: EOS, Tronix, VeChain, OmiseGO, MOD Token
- **ERC20 Token (Simplified)**
  - No allowance / approval / transferFrom, also no Approval event
  - Using SafeMath (always use it)
  - Creator gets 1000 coins

```
contract SimpleERC20 {
    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns
(uint256);
    function transfer(address to, uint256 value) public
returns (bool);
    event Transfer(address indexed from, address indexed to,
uint256 value);
}

string public constant name = "VSS-TOKEN";
    string public constant symbol = "VST";
    uint8 public constant decimals = 18;

    using SafeMath for uint256;
    mapping(address => uint256) balances;
    uint256 totalSupply_;

    constructor() public {
        totalSupply_ = 1000 * (10**18);
        balances[msg.sender] = totalSupply_;
    }
…
```

ERC20 - Considerations

- **Transfer ownership**
  - Important for minting: after minting, set new owner
  - Private key never on the minting machine
- **Token lockups**
  - Vest tokens – make sure big investor don’t dump
  - Team vesting – show the world that you believe in your token
- **Minting**
  - Lock contract during minting

```
function transferOwnership(address _newOwner)
public {
        require(owner == msg.sender);
        owner = _newOwner;
}
```

```
mapping(address => uint256) lockups;
…
if (lockups[msg.sender] != 0) {
            require(now >= lockups[msg.sender]);
}
```

```
bool public mintingDone = false;
…
require(mintingDone == true);
```

ERC20 - Considerations

- **Utility Token – ERC 223 / 677**
  - Eliminates the problem of lost tokens
    - QTUM, $1,204,273 lost
    - EOS, $1,015,131 lost
  - Allows developers to handle incoming token transactions
  - Backwards compatible?
    - ERC 223: throw if transferring to a contract that does not implement tokenFallback… (corner cases)
  - ERC 677 fully backwards compatible, but tokens still can be lost
- **transferAndCall()**
  - Implement token now, define utility later

```
contract ERC677 {
    event Transfer(address indexed _from, address indexed _to, uint256 _value,
bytes _data);
    function transferAndCall(address _to, uint _value, bytes _data) public
returns (bool success);
}

contract ERC677Receiver {
    function tokenFallback(address _from, uint _value, bytes _data) public;
}

// ERC677 functionality
function transferAndCall(address _to, uint _value, bytes _data) public returns
(bool) {
require(transfer(_to, _value));
    …
   if (isContract(_to)) {
            ERC677Receiver receiver = ERC677Receiver(_to);
            receiver.tokenFallback(msg.sender, _value, _data);
}
//ERC223
function transfer(address _to, uint _value) public returns (bool success) {
    bytes memory empty;
    if(isContract(_to)) {
        return transferToContract(_to, _value, empty);
    }
    else {
        return transferToAddress(_to, _value, empty);
    }
}
```

URL: b.socrative.com/student/

Room: HSR

Security Considerations

- **List of the top smart contract vulnerabilities**
  1. Reentrancy (DAO 3.5M, 50M)
  2. Access Control (Parity 150K, 30M)
  3. Arithmetic Issues (DAO)

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

Contract AA {
function attack() {
        A a = A(addressOfA);
        a.withdraw(100);
}
function () payable {
        A a = A(addressOfA);
        a.withdraw(100);
}
}
function initContract() public {
        owner = msg.sender;
}

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

Security Considerations

- **List of the top smart contract vulnerabilities**
  1. Reentrancy (DAO 3.5M, 50M)
  2. Access Control (Parity 150K, 30M)
  3. Arithmetic Issues (DAO)
  4. Unchecked Return Values For Low Level Calls
  5. Denial of Service (Parity 500K, 300M)
  6. Bad Randomness (400K)
     - Future blockhash as random source, but miner can influence it.

```
function withdraw(uint256 _amount) public {
        require(balances[msg.sender] >= _amount);
        balances[msg.sender] -= _amount;
        etherLeft -= _amount;
        msg.sender.send(_amount);
}

function getEtherBalanceIndex(address _address) internal
returns (uint256) {
       for (uint256 i = 0; i < investors.length; i++) {
           if (investors[i] == _address) {
               return i;
           }
       }
       return investors.length;
   }

function play() public payable {
        require(msg.value >= 1 ether);
        if (block.blockhash(blockNumber) % 2 == 0) {
                msg.sender.transfer(this.balance);
        }
}
```

Security Considerations

- **List of the top smart contract vulnerabilities**
  1. Reentrancy (DAO 3.5M, 50M)
  2. Access Control (Parity 150K, 30M)
  3. Arithmetic Issues (DAO)
  4. Unchecked Return Values For Low Level Calls
  5. Denial of Service (Parity 500K, 300M)
  6. Bad Randomness (400K)
  7. Front-Running
  8. Time Manipulation - A malicious miner sets own timestamp ~ 900s
  9. Short Address Attack – not yet exploited
  10. Unknown Unknowns

1. A smart contract publishes an RSA number (N = prime1 x prime2).
2. A call to its submitSolution() public function with the right prime1 and prime2 rewards the caller.
3. Alice successfully factors the RSA number and submits a solution.
4. Someone on the network sees Alice's transaction (containing the solution) waiting to be mined and submits it with a higher gas price.
5. The second transaction gets picked up first by miners due to the higher paid fee. The attacker wins the prize.

We believe more security audits or more tests would have made no difference. The main problem was that reviewers did not know what to look for.

Christoph Jentzsch

Security Considerations

- **Multiple Exchanges Suspend ERC20 Token Trading Due To Potential BatchOverflow Bug**
  - Integer overflows in multiple contracts
  - Poloniex suspended all ERC20 token deposits and withdrawals,
  - OKEX's decision to halt ERC20 deposits

```
255   function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
256     uint cnt = _receivers.length;
257     uint256 amount = uint256(cnt) * _value;
258     require(cnt > 0 && cnt <= 20);
259     require(_value > 0 && balances[msg.sender] >= amount);
260
261     balances[msg.sender] = balances[msg.sender].sub(amount);
262     for (uint i = 0; i < cnt; i++) {
263         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
264         Transfer(msg.sender, _receivers[i], _value);
265     }
266     return true;
267   }
268 }
```

URL: b.socrative.com/student/

Room: HSR

ERC20 - Considerations

- **Minting done in batches**
  - Around 100/150 until max gas used
- **Only owner can mint**
  - Once minting finished no one can ever mint
- **Check maxSupply**
  - Don’t mint more tokens, check important, as minter needs to respect limits
- **Emit from:0**
  - Discussion if from is 0 or contract address

```
function mint(address[] _recipients, uint256[] _amounts)
public {
        require(owner == msg.sender);
        require(mintingDone == false);
        require(_recipients.length == _amounts.length);
        require(_recipients.length <= 256);

        for (uint8 i = 0; i < _recipients.length; i++) {
            address recipient = _recipients[i];
            uint256 amount = _amounts[i];

            // enforce maximum token supply
            require(totalSupply_.add(amount) <= maxSupply);

            balances[recipient] =
balances[recipient].add(amount);
            totalSupply_ = totalSupply_.add(amount);

            emit Transfer(0, recipient, amount);
        }
}
```

ERC20 - Considerations

- **ERC865: Pay transfers in tokens instead of gas, in one transaction**
  - Delegate transfer of tokens to a third party
  - UX: pay service in tokens – user needs to have ETH **and** tokens
    - Create account at exchange, KYC, Bank transfer – and wait
- **Client could be offchain, third party not**
  - Make sure the signature is unique
- **ERC677**
  - Add delegatedTransferAndCall()

```
function delegatedTransfer(
          uint256 _nonce,
          address _from,
          address _to,
          uint256 _value,
          uint256 _fee,
          uint8 _v,
          bytes32 _r,
          bytes32 _s) public returns (bool) {

uint256 total = _value.add(_fee);
require(_from != address(0));
require(_to != address(0));
require(total <= balances[_from]);
require(_nonce > nonces[_from]);

address delegate = msg.sender;
address token = address(this);
bytes32 delegatedTxnHash = keccak256(delegate, token, _nonce, _from, _to,
_value, _fee);
address signatory = ecrecover(delegatedTxnHash, _v, _r, _s);
require(signatory == _from);

balances[_from] = balances[_from].sub(total);
balances[_to] = balances[_to].add(_value);
balances[delegate] = balances[delegate].add(_fee);
nonces[_from] = _nonce;

DelegatedTransfer(_from, _to, delegate, value, fee);
return true;
}
```

EVM Considerations - Random Numbers

- **Random Numbers**
  - There is no random number in Ethereum
  - Every EVM needs to come to the same result
- **Random Numbers from Oracle (External source)**
  - Or: commitment schemes
- **Or: use future blockhash**
  - Only up to 256 blockhashes from the past can be accessed.
  - Deduct / add tokens/ETH from the past address
  - Miner can influence the random value in a sense
    - Value needs to be low (miner gets 2 ETH)

```
function transfer(address _to, uint256 _value) public returns (bool) {
     …
        //since this is a lucky coin, the value transferred is not what you
expect
        luckyTransfer();
        previousTransferBlockNr = block.number;
        previousTransferAddress = msg.sender;
        …
        uint256 val = _value.sub(potIncrease);
       pot = pot.add(potIncrease);
}

function luckyTransfer() private {
        if(block.number != previousTransferBlockNr && (block.number -
previousTransferBlockNr) < 256 ) {
            uint256 rnd =
uint256(keccak256(block.blockhash(previousTransferBlockNr)));
            if(rnd % 200 == 0) { //.5%
                balances[previousTransferAddress] =
balances[previousTransferAddress].add(pot);
                Transfer(this, previousTransferAddress, pot);
                //tokens are from pot, thus no tokens are created from thin air
                pot = 0;
                previousTransferBlockNr = 0;
                previousTransferAddress = 0;
            }
        }
}
```

URL: b.socrative.com/student/

Room: HSR

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch