香港城市大學
City University of Hong Kong

CS 6290

Privacy-enhancing Technologies

Department of Computer Science

Slides credit in part from E. Ben-Sasson, D. Dziembowski, I. Eyal, R. Geambasu, and F. Greenspan, A. Judmayer, A. Juels, A. Miller, J. Poon, V. Shmatikov, D. Song, and F. Zhang

Lecture 4 –Smart Contract Applications and Security

CS Department
City University of Hong Kong

Ethereum Smart Contract

Designing

Key Challenges in Smart Contract Design

- Smart Contracts in Ethereum can be trusted for *correctness* and *availability*, but not **privacy**
- Blockchain resources are expensive
- Race conditions and temporary forks
  - *You need to be considerate*
- Obscure and counterintuitive VM rules
  - Programming in Ethereum is not developer-friendly yet

Elements of A Smart Contract Applications

- The smart contract itself
- Client code for the parties in the contract
- Protocol narrations
  - What happens in the *ordinary case*, where all parties are honest?
  - What are the attacks / edge cases that your smart contract defends against?
  - What are attacks that your smart contract *does not* defend against?

Application I:
Rock Paper Scissors

using Hash Commitments and Collateral

Rock Paper Scissors

Intended behavior:

- Any two parties sign up to play, deposit $1
- Each party chooses Rock, Paper, Scissors
- Both parties reveal their choice
- Winner gets $2

Threat model: the other party is **malicious**

Goals: an honest party is guaranteed an equal or better payoff distribution

Problem 1: Race conditions

- What happens when 3 players try to sign up at once?
  - Which two players are involved
  - How to handle the deposit of the remaining player
- You need to specify a rule for the contract to follow!
- Or, this design flaw on the contract might cause *money leaks*
  - What if we send more than $1000?

Problem 2: Front running!

- Whoever goes second in the game can always win
  - The front runner's choice is leaked on the blockchain
  - The Ethereum blockchain is publicly visible

- Record hash commitment of the choice on the blockchain instead!
  - The player goes second cannot learn the choice
  - Reveal the choice once both two players have committed their choices

Problem 3: Fairness!

- Whoever reveals second in the game can quit
  - I'm going to flee because I'm losing
  - You can never trace me because I'm just no-one with a pseudonymous address

- Solution: collateral deposits
  - I will have your deposited money instead!
  - But how much is suitable for advance deposit?
    - Game theory [CCS' 17]

Application II: Utility Token for Games

Booming Token Economy

- Economic systems facilitated or executed with tokens
- Decisions are made through token supply & demand
- Participants act upon incentives
- Big names: Bitcoin, Ethereum, Ripple, EOS, IOTA, OmiseGO, Maker, Filecoin …

Security Token vs. Utility Token

- Security token
  - Ownership rights or share of a company
  - Subject to stricter regulation requirements
- Utility token
  - For dedicated service access
  - Currently less regulation is imposed
- No definite distinction between the two

Filecoin

golem

omisego

Our narrative – a utility token for games

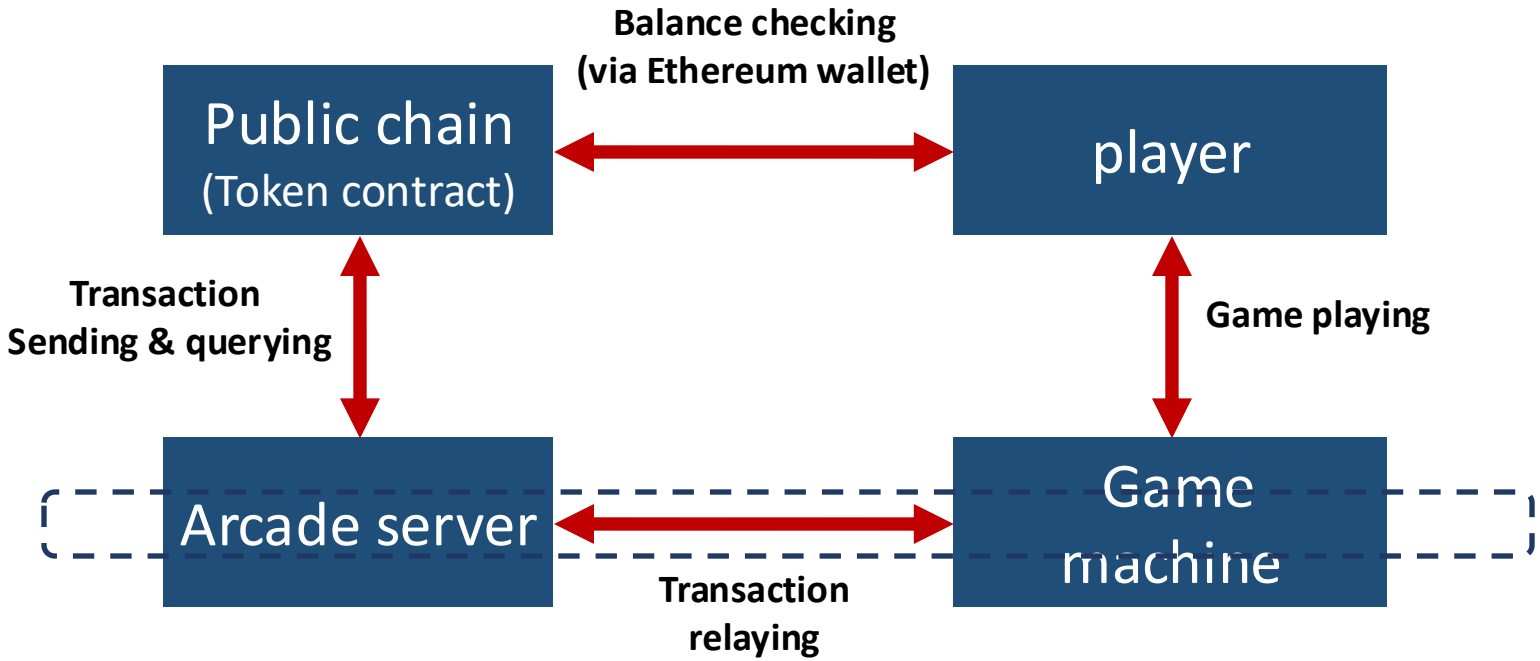
- Reviving game arcade
  - Retain existing players
  - Attract new players
- Use token to unlock barriers in the past
  - Arcade operators may attract players from wider demographic range
  - Players may have more gaming choices with token rewards
  - Potentially attract more game developers to join this tokenized model

Our narrative – a utility token for games

- Players get token rewards while playing
  - Every X rounds
  - Every X achievement
  - Every X “coin” in
- Players consume tokens to enhance gaming experience
  - In-game level up
  - In-game power boost
  - Special cosmetic badges
  - Unique themes
  - More in-game stash
  - …

Basic version purely based on public chain

- Architecture



Assumptions

- The game arcade is trusted
  - Game machine has a private key built in
  - Game machine can sign transaction with the private key
  - The arcade has a local server to relay transactions from machines

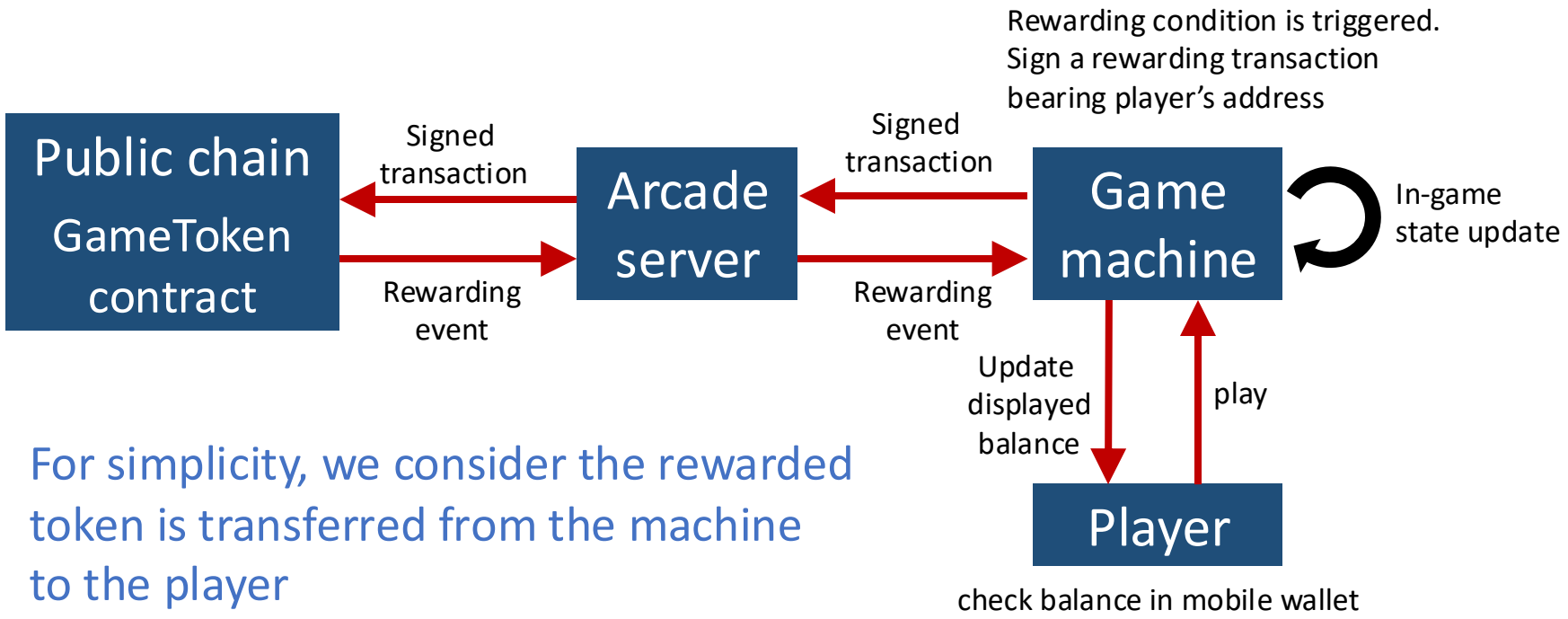
Assumptions

- We do not consider the performance/cost issues
  - TX delay – how long it takes for the transaction result to return
  - TX rate – how frequent the machine generate transaction
  - TX fee – how much it costs the arcade to run the system

Our Contract Design

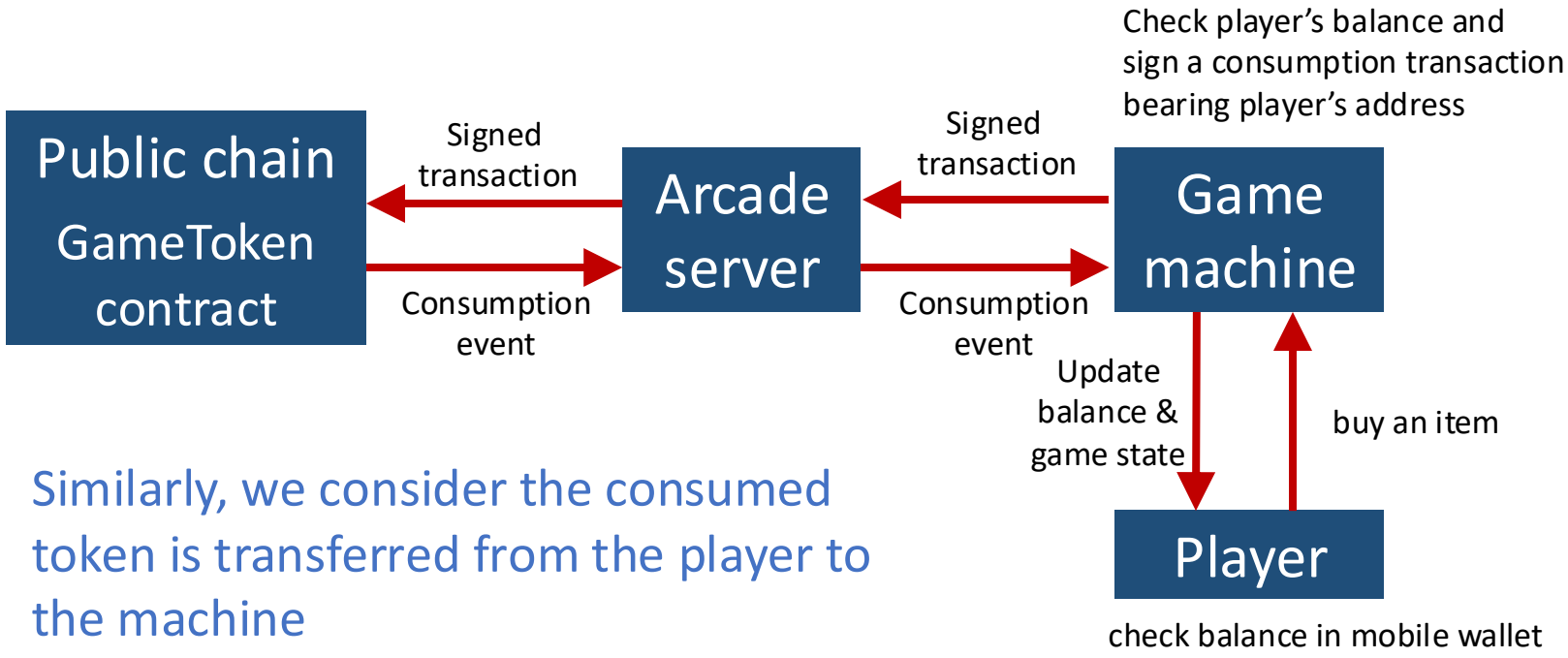
- ERC20 token
  - Other standards: ERC223, ERC621, ERC827,etc.
- Token rewarding
  - Authorized game machines can reward players
- Token consumption
  - Authorized game machines can consume token for players to enhance gaming experience

Token rewarding



For simplicity, we consider the rewarded token is transferred from the machine to the player

Token consumption



Similarly, we consider the consumed token is transferred from the player to the machine

Reward and consume function in contract

- events, data fields and modifiers

```
event Reward(address indexed machine, address indexed player, uint256 value);
event Consume(address indexed machine, address indexed player, uint256 value);

mapping(address => bool) public _authorizdedMachines;

modifier onlyOwner()  {
    require(msg.sender == _owner);
    _;
}

modifier onlyAuthorizedMachine()  {
    require(_authorizdedMachines[msg.sender]);
    _;
}
```

Reward and consume function in contract

- access control

```
function addGameMachine(address machine) public onlyOwner() {
    _authorizdedMachines[machine] = true;
}

function removeGameMachine(address machine) public onlyOwner() {
    _authorizdedMachines[machine] = false;
}
```

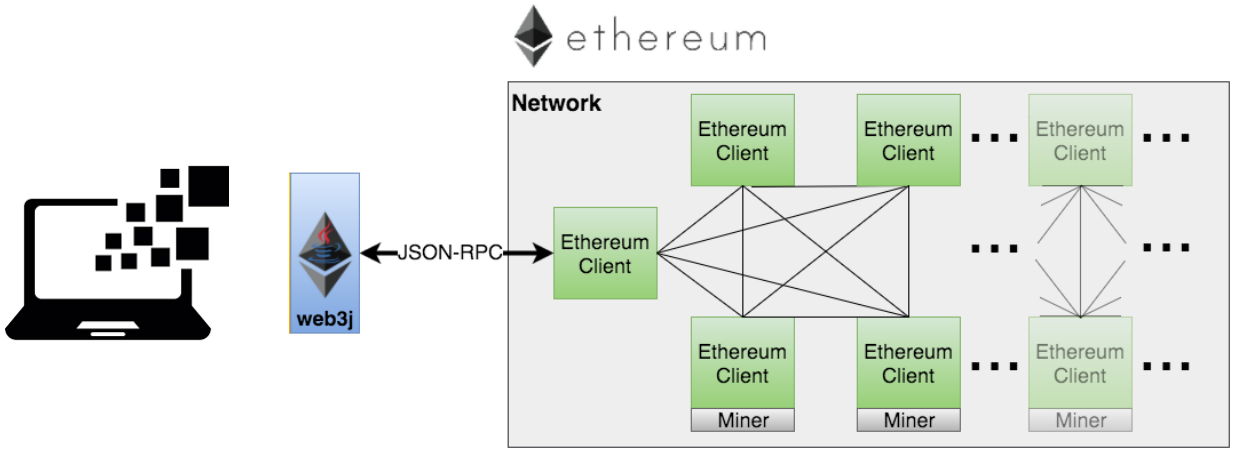
Reward and consume function in contract

- reward and consume

```
function reward(address to, uint256 value) public onlyAuthorizedMachine() returns (bool) {
    _transfer(msg.sender, to, value);
    emit Reward(msg.sender, to, value);
    return true;
}

function consume(address by, uint256 value) public onlyAuthorizedMachine returns (bool){
    _transfer(by, msg.sender, value);
    emit Consume(msg.sender, by, value);
    return true;
}
```

From contract to blockchain application

- Smart contracts only live on blockchain
- An outside application can interact with them via a set of API
- Ethereum JSON-RPC API



From contract to blockchain application

- Example API

eth_getBalance - Returns the balance of the account of given address.

Parameters

```
params: [
   '0xc94770007dda54cF92009BFF0dE90c06F603a09f',
   'latest'
]
```

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBalance","params":["0xc9477000

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x0234c8a3397aab58" // 158972490234375000
}
```

eth_sendTransaction - Creates new message call transaction or a contract creation

Parameters

```
params: [{
  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
  "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
  "gas": "0x76c0", // 30400
  "gasPrice": "0x9184e72a000", // 10000000000000
  "value": "0x9184e72a", // 2441406250
  "data": "0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb9
}]
```

Example

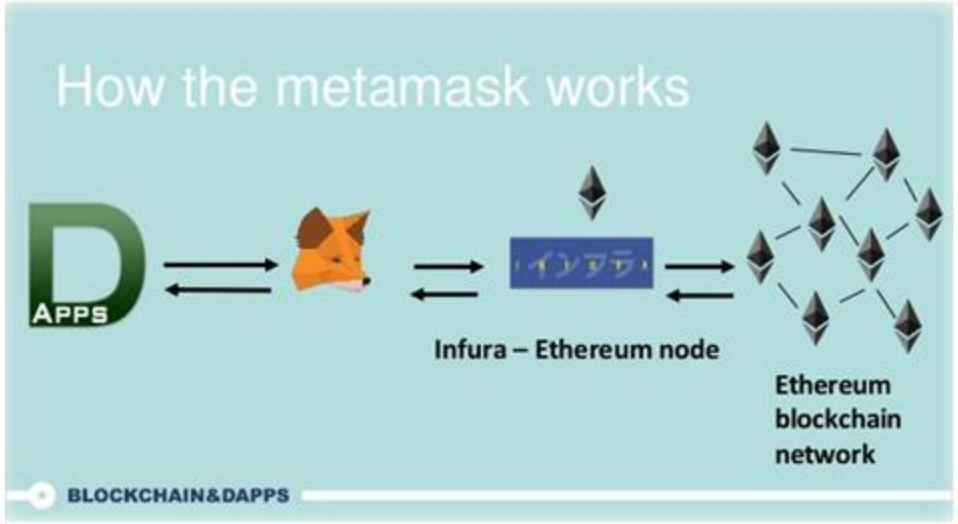
```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_sendTransaction","params":[{see a

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

From contract to blockchain application

- The API requests (essentially transactions) are handled by a blockchain node, which for Ethereum runs a client like Geth or Parity
- We need to **trust the node** for
  - Handling and relaying request faithfully
  - Broadcasting the transactions to the network
  - Notifying us the events emitted from the transactions
- Alternatively, one can broadcast transactions to multiple nodes that are not necessarily fully trusted

From contract to blockchain application

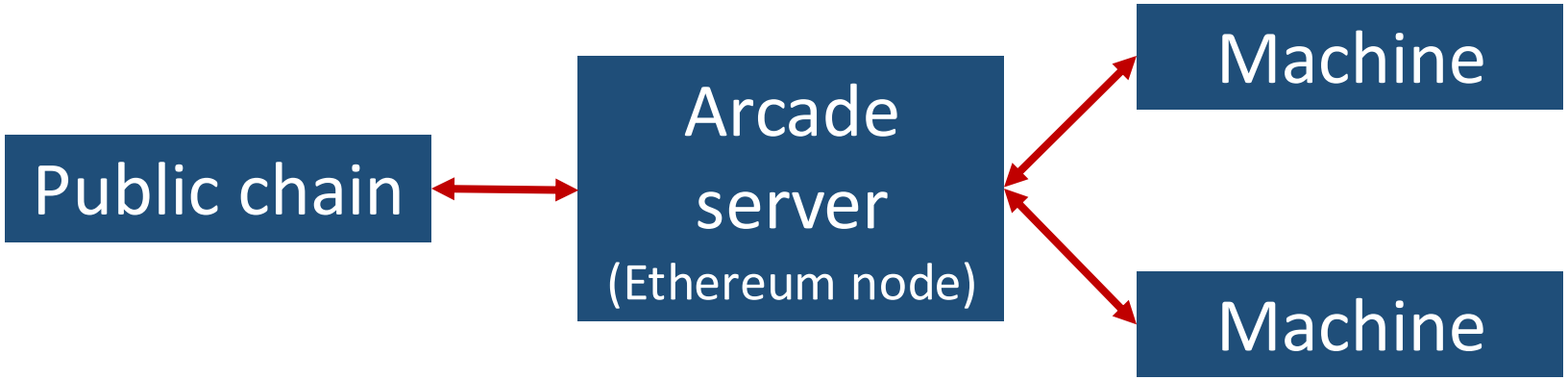
- In previous hands-on session, the transactions for deploying contracts are relayed by MetaMask to a public Ethereum node cluster (Infura).



https://www.slideshare.net/Kenneth_hu/20180711-metamask

From contract to blockchain application

- In our case, the game arcade server will run a local Ethereum node



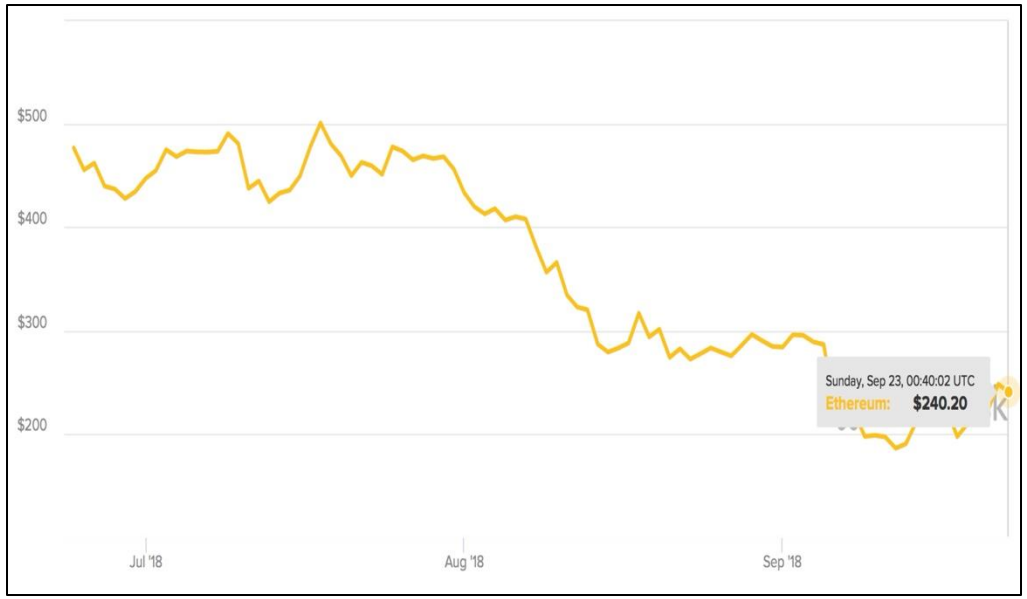
When deployed to the real Ethereum mainnet

- **Transaction speed** matters
  - Players desire the token to take effect instantaneously
- **Transaction rate** is also a concern
  - Multiple machines can reward and consume at the same time
- **Transaction fee** should be minimized
  - Game arcades don't want high operation cost for the token system

Public chain alone is not enough

A single DApp can bring down the whole network

ETH price is still too high for daily operation



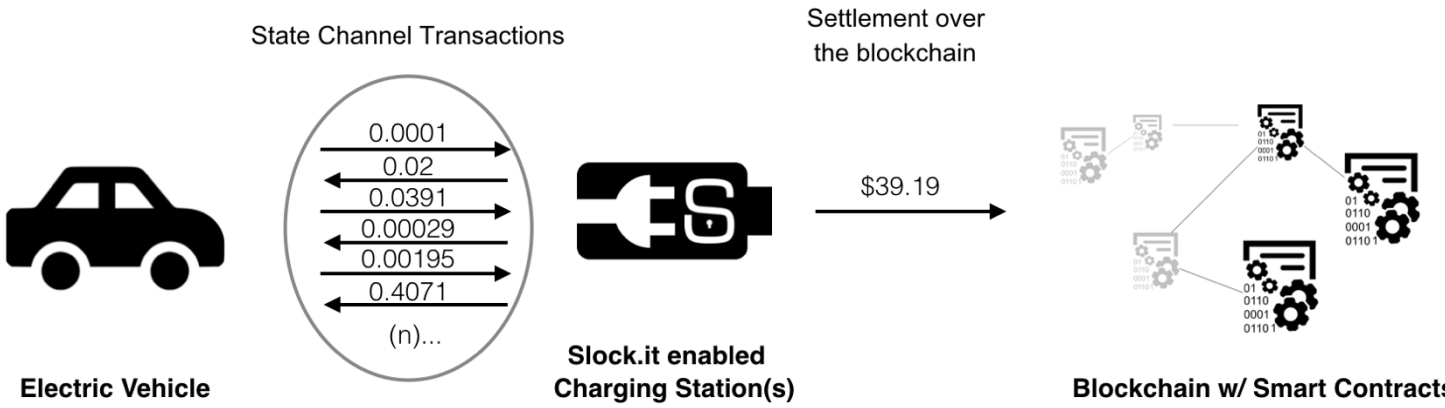
https://www.coindesk.com/ethereum-price/

How to solve the performance and cost issue?

- This is generally referred to as blockchain *scalability* problem
- General directions
  - Layer 1 – Sharding
    - Redesign base-layer blockchain protocol
    - Each TX is only seen and processed by a small shard of nodes
    - Examples: Zilliqa, Prysmatic Labs Ethereum 2.0
  - Layer 2
    - State channels
    - Sidechains (aka parachains, childchains)

Scaling technique – state channels

- **One-to-one channel** for bulk off-chain transactions
  - Final settlement on-chain
- Example: Lightning Network, Raiden Network, Counterfactual

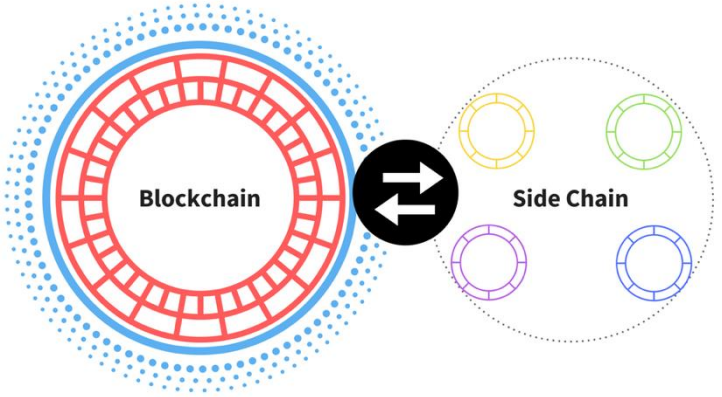


https://cdn-images-1.medium.com/max/2000/1*YjHwvs76f1-CgGx43SwvEw.png

Scaling technique – sidechains

- A **separate chain** running in parallel to the main chain
  - Independent consensus (e.g., PoA) and configurations
  - Data/states can be exchanged between the two chains
- Examples: Plasma, POA Network, Parity Bridge, (Polkadot, Cosmos)

We follow this direction!



https://cdn-images-1.medium.com/max/1600/0*gHDyU65BfvNG-VHn.png

Solution: consortium chain

- Game arcades maintain an Ethereum-compatible consortium chain
  - We will bridge the consortium chain with the public chain
- Rationale
  - We can reuse previously developed components
    - GameToken contract
    - Server application
  - More flexibility in upgrading and adding new features
  - Different games arcades can naturally form a consortium

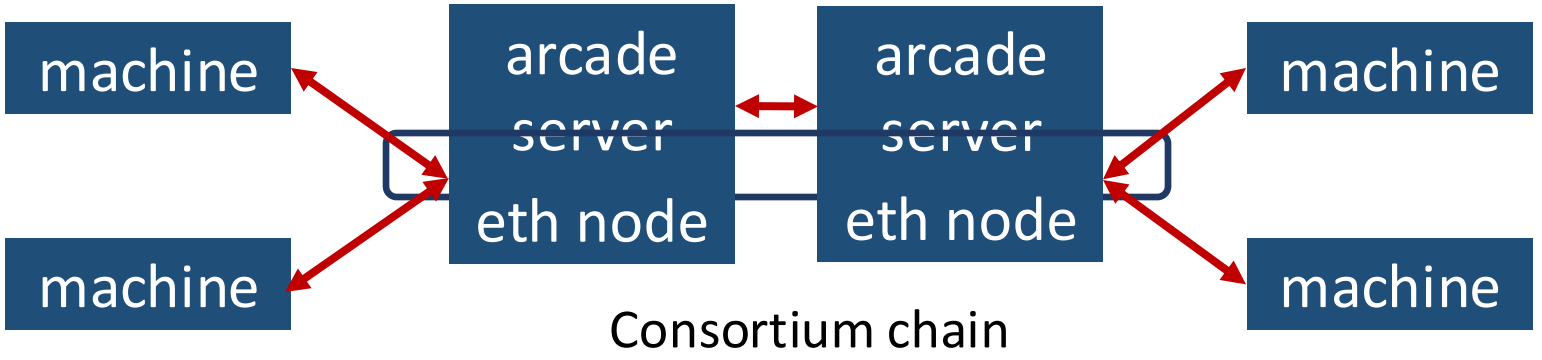
Solution: consortium chain

- The consortium chain can be configured with
  - low TX delay – e.g., 1 second block interval
  - high TX rate – e.g., unlimited block gas limit
  - zero TX fee – e.g., zero transaction gas price



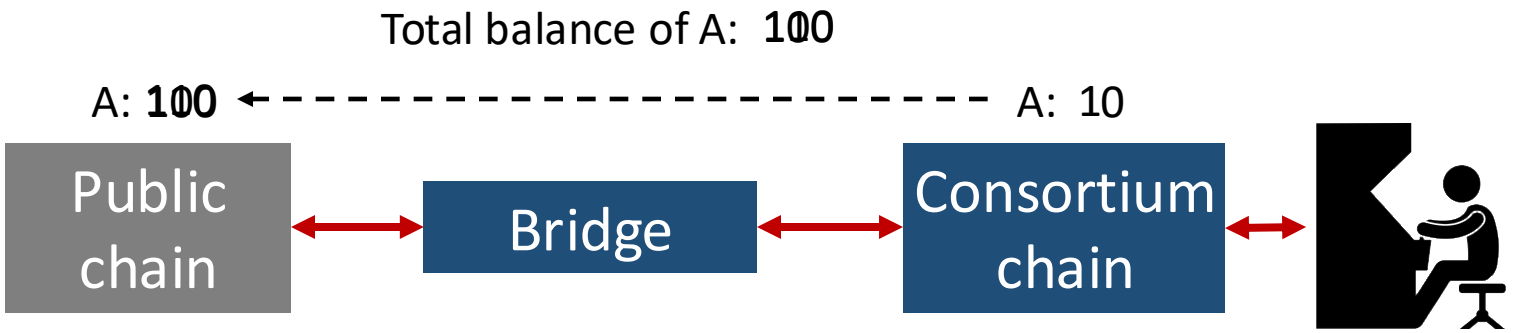
Solution: consortium chain

- Game machines only interact with the consortium chain
- Token rewarding and consumption only occur on consortium chain



Solution: consortium chain + cross-chain bridge

- Players can exchange tokens between the two chains via the bridge
  - "Burn" on one chain, "mint" on the other

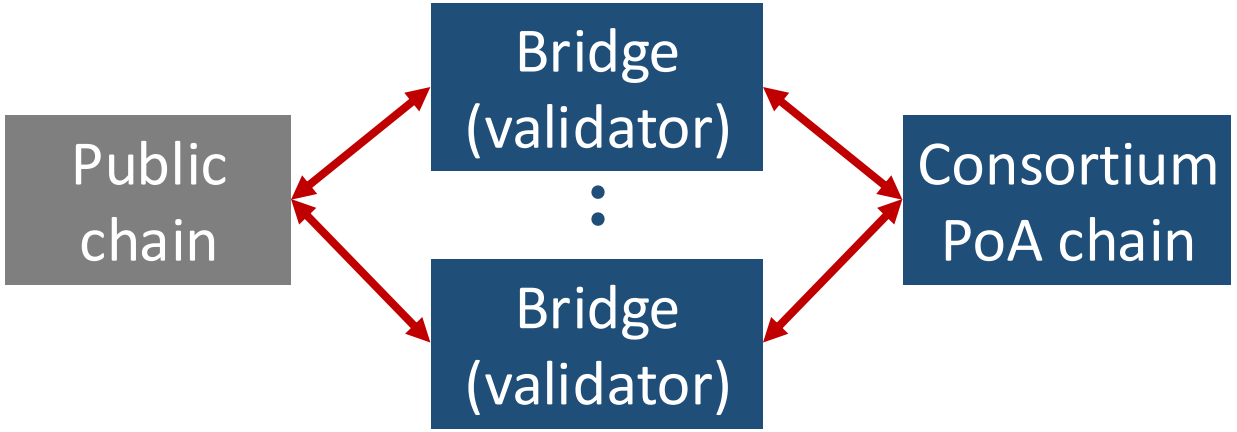


Solution: consortium chain + cross-chain bridge

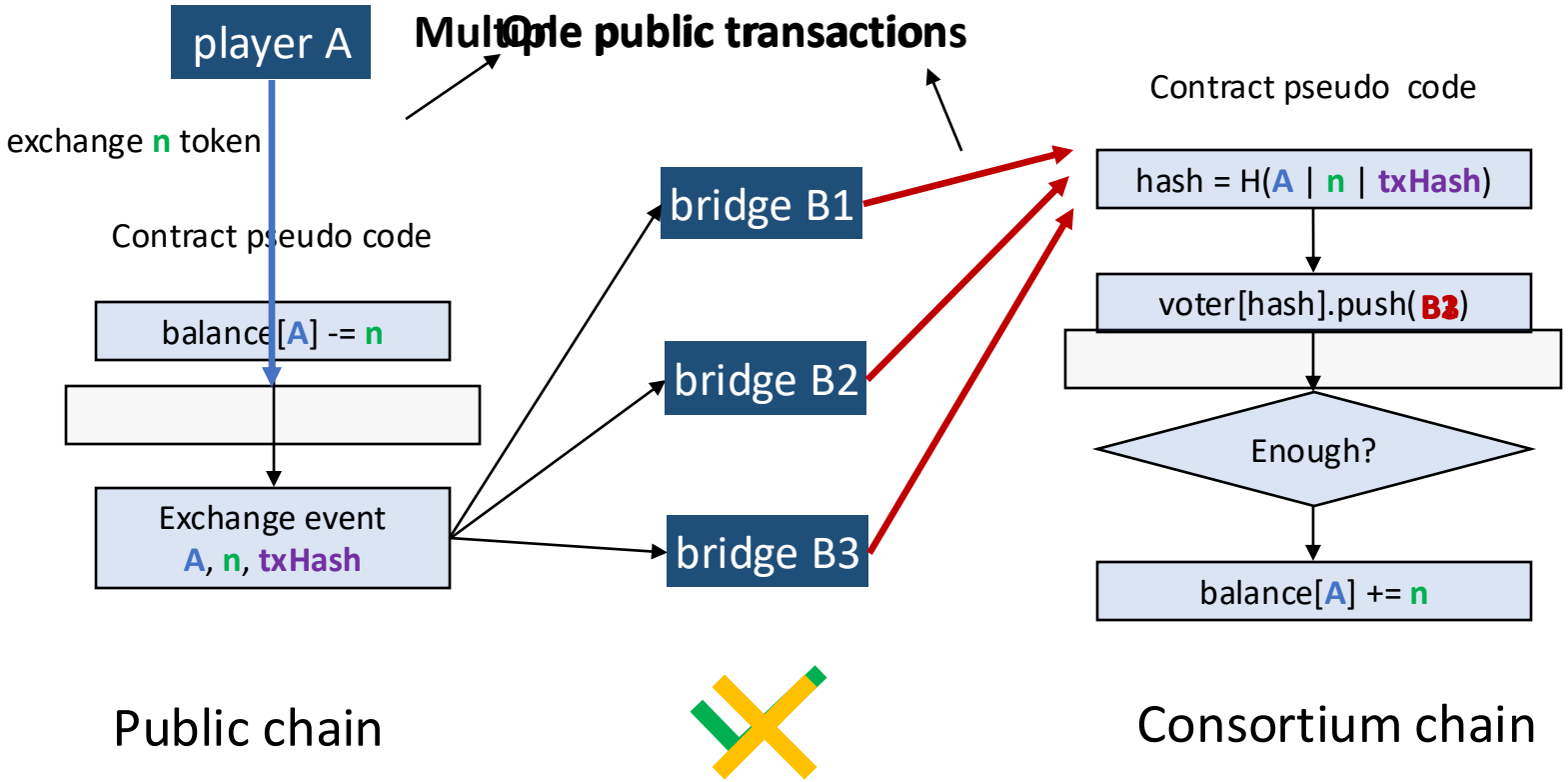
- We use Proof-of-Authority (PoA) consortium chain
  - Only a set of **validators** can propose and write blocks
  - Each game arcade can run a validator
  - Wrongdoing validators can be voted out
  - Less de-centralized than PoW, PoS
- Example PoA consensus algorithms
  - AuthorityRound (Aura)
  - Clique
  - Tendermint

Solution: consortium chain + cross-chain bridge

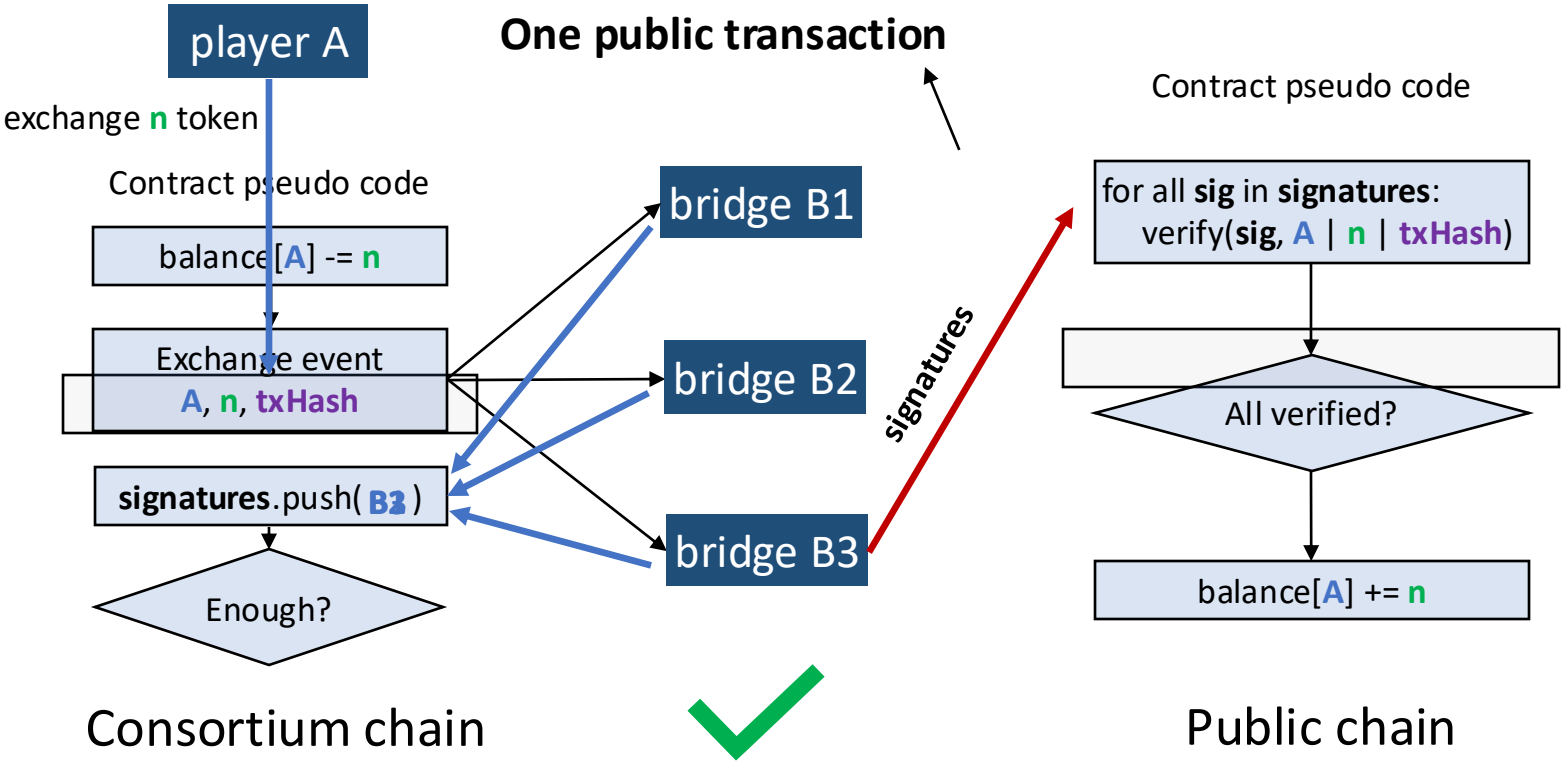
- Each validator additionally plays the role of a bridge
  - Cross-chain state transfer requires **multisig** from bridges
  - An exchange is approved when a required number of bridges have signed



Multi-sig bridging – the straightforward way



Multi-sig bridging – the more economic way



Already implemented and deployed by

Secure Computation with Blockchain

Privacy-preserving smart contract over encrypted data

Exemplary applications

- Contract for knowledge discovery + monetization
  - Aggregate **private** inputs from crowd participants
  - Issue rewards (cryptocurrencies) to the participants
- Contract for E-voting
  - Tally **private** votes from crowd voters.

Practical requirements

- Confidentiality of data throughout contract processing
- Validity of data in confidential transactions
  - E.g., for numerical data processing (voting, statistics aggregation), data must be in a valid range
- Affordable on-chain monetary cost

Ideal-case solution

- Blockchain conducts all the secure procedures
  1. Clients submit the encrypted data and validity proof to blockchain
  2. Blockchain checks the validity proofs
  3. Blockchain performs aggregation over all encrypted data

Can we realize all operations on current blockchain platforms?

Existing practices are not sufficient

- Most existing solutions focus on private monetary transfer
  - Monetary values sealed in a transaction are within valid range
    - Monero -- Bulletproofs
    - Zcash -- zk-SNARK
    - BTCP (Bitcoin Private) -- zk-SNARK

- NOT immediately available for general privacy-preserving smart contract over encrypted data
  - Encrypted data and validity proof + on-chain execution logic = guaranteed (encrypted) result

Moreover, current EVM has many constraints

- Lack of necessary support on many cryptographic operations in Solidity and EVM. For example,
  - No zero knowledge proof verification functionalities
  - No advanced encryptions support
  - Lacking developer-friendly libraries
  - …
- Also, on-chain operations and storage can be expensive

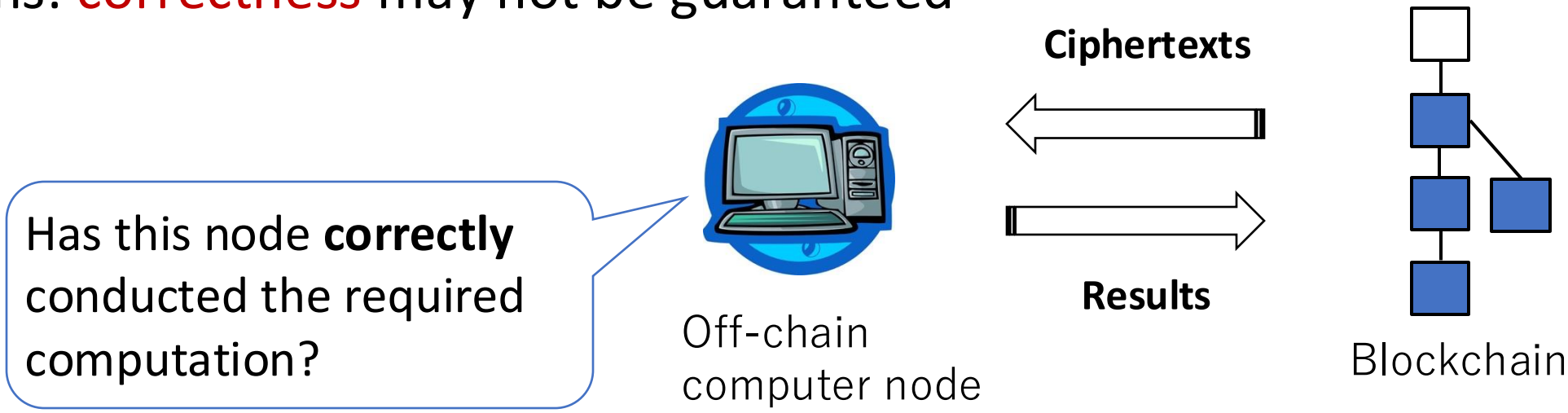
What can we do?

- Wait for the EVM to evolve, or switch to alternative platforms
  - e.g., EOS that supports C/C++

- Can we move some processing off-chain with result verification?
  - Pros: leverage a wider range of tools/libraries, e.g., bulletproof and lib-snark
  - Cons: off-chain processing node(s) can be dishonest
    - **How to verify off-chain results and record it back on-chain?**

Off-chain computation

- Download on-chain submissions, conduct the checking and computation via off-chain node(s), and return the result back
- Pros: scalability and cost-efficiency
- Cons: correctness may not be guaranteed



Smart contract verify off-chain computation?

- Back to the dilemma where current EVM is still lack of necessary support on many cryptographic operations
- Also, no practical solutions for general verifiable computations

More practical choices ?

- Elect a set of off-chain nodes (committee)
  - Each off-chain node verify the proof then cast a vote if its local verification process passes
  - Trust assumption? fixed nodes or select on-the-fly?

- Trusted hardware, e.g., Intel SGX
  - Rely on the trust of hardware manufactures
  - Trusted execution result

Example: SGX-enabled solution

- Contract **now** only
  a. records clients' submissions;
  b. guarantees that the result is returned from SGX, i.e., via wallet signature.

SGX enclave

```
Contract Agg_blockchain{

\\ validate a client's submission
function data_submission {

Receive encrypted data and range proof from a client

Check the range proof

if the data is valid

        accept the encrypted data     }

\\ aggregate after a time slot when all clients have submitted
function aggregation {

Retrieve all accepted encrypted data

Perform secure computation

Return the result     }

}
```

Smart Contract Security Case Studies

Smart contract security in general

- Secure development practices
- Formal verification
- Safer type systems and languages
- Code auditing
- Bug bounty

Some special functions in Solidity

- **fallback** function
  - No name, no argument, no return, external visibility
  - Executed if no explicit function of the contract is called
  - Can be marked payable to receive plain Ether
- **call** and **delegatecall** – low level function calls
  - **call**: executed in the context of callee contract
  - **delegatecall**: executed in the context of caller contract

Case 1: use functions from a fancy library

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(uint)", param);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

GameToken contract

```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fancy token library contract

Case 1: use functions from a fancy library

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(uint)", param);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

GameToken contract

```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fancy token library contract

Case 1: use functions from a fancy library

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(uint)", param);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

GameToken contract

```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fancy token library contract

Case 1: use functions from a fancy library

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(uint)", param);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

GameToken contract

The real function can be specified in msg.data

```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fancy token library contract

Case 1: initialization only once?

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(uint)", param);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

GameToken contract

```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fancy token library contract

Case 1: an attacker can take the ownership

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(ui
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

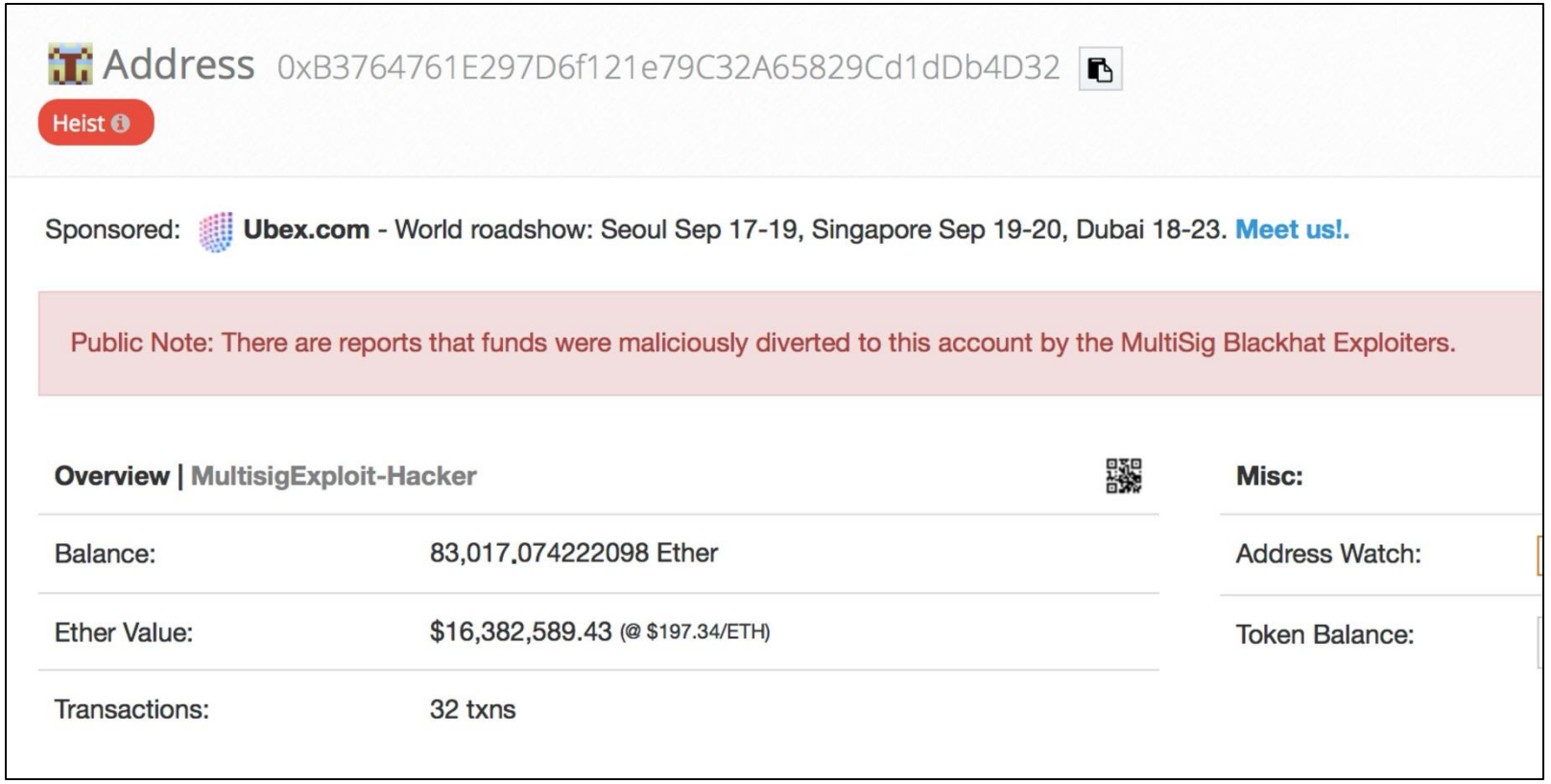
Game token contract


```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fancy token library contract

Case 1: the ramification and a real attack

- The infamous Parity Multisig Exploit - **Wallet** using **WalletLibrary**
  - The attacker immediately earns 26,793 ETH (~6.1M USD back then)



Case 1: a quick fix

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;
    bool initialized;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(uint)", param);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

Fixed game token contract

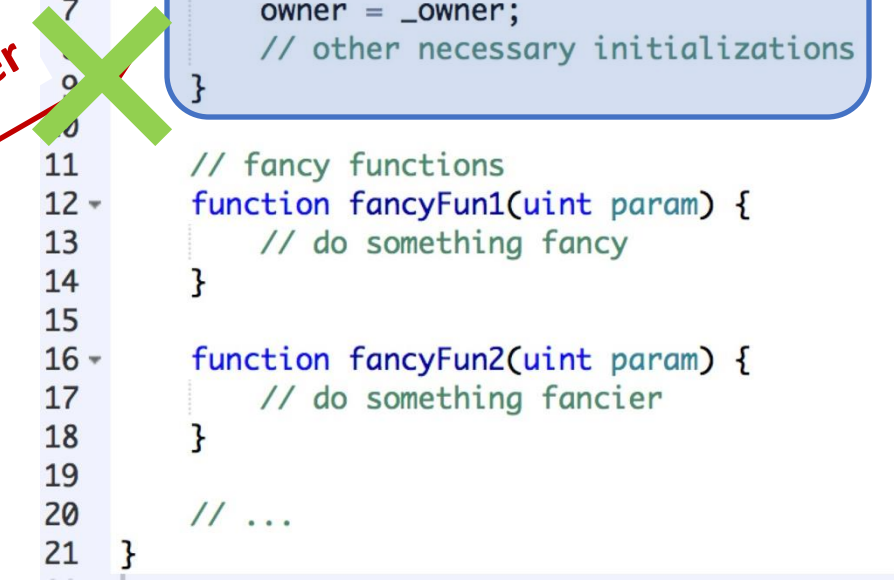
```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        require(initialized == false);
        initialized = true;
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fixed fancy token library contract

Case 1: a quick fix

```solidity
pragma solidity ^0.4.24;

contract GameToken {
    address fancyTokenLib;
    address owner;
    bool initialized;

    constructor (address _owner) {
        fancyTokenLib = 0x1234567890ABCDEF1234567890ABCDEF12345678;
        fancyTokenLib.delegatecall( "initToken(address)", _owner);
        // do other initialization
    }

    // some advanced functions from fancyTokenLib
    function fancyFun1(uint param) {
        fancyTokenLib.delegatecall( "fancyFun1(
                                    am);
    }

    // fallback function
    function () payable {
        fancyTokenLib.delegatecall(msg.data);
    }

    // implementation of ERC20 interface
    // ...
}
```

Fixed game token contract




```solidity
contract fancyTokenLib {
    address owner;

    function initToken(address _owner) {
        require(initialized == false);
        initialized = true;
        owner = _owner;
        // other necessary initializations
    }

    // fancy functions
    function fancyFun1(uint param) {
        // do something fancy
    }

    function fancyFun2(uint param) {
        // do something fancier
    }

    // ...
}
```

Fixed fancy token library contract

Case 1: a sequel “accident” of Parity Multisig

- The **WalletLibrary** itself is a contract, so its “owner” can be “set”
- If **WalletLibrary** is left uninitialized, then an attacker can take its ownership

anyone can kill your contract #6995

Open **devops199** opened this issue 22 hours ago · 12 comments

devops199 commented 22 hours ago • edited

I accidentally killed it.

https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

Hello, first of all i'm not the owner of that contract. I was able to make myself the owner of that contract because its uninitialized.

These (https://pastebin.com/ejakDR1f) multi_sig wallets deployed using Parity were using the library located at "0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4" address. I made myself the owner of "0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4" contract and killed it and now when i query the dependent contracts "isowner(<any_addr>)" they all return TRUE because the delegate call made to a died contract.

I believe some one might exploit.

The WalletLibrary contract

Slide credit to Andrew Miller

anyone can kill your contract #6995

Parity wallet uses “Prototype Inheritance”
The WalletLibrary should only act as a template.
- It should not have any “state”
- Its constructor has never been called

But, the WalletLibrary is actually just a contract!

Attacker calls:

```
library.call(“initWallet(address)”,   attacker);
library.kill();
```

Thereafter, any method call to any instance “Wallet” will fail, since the target of delegatecall is destroyed

Wallet Library contract:

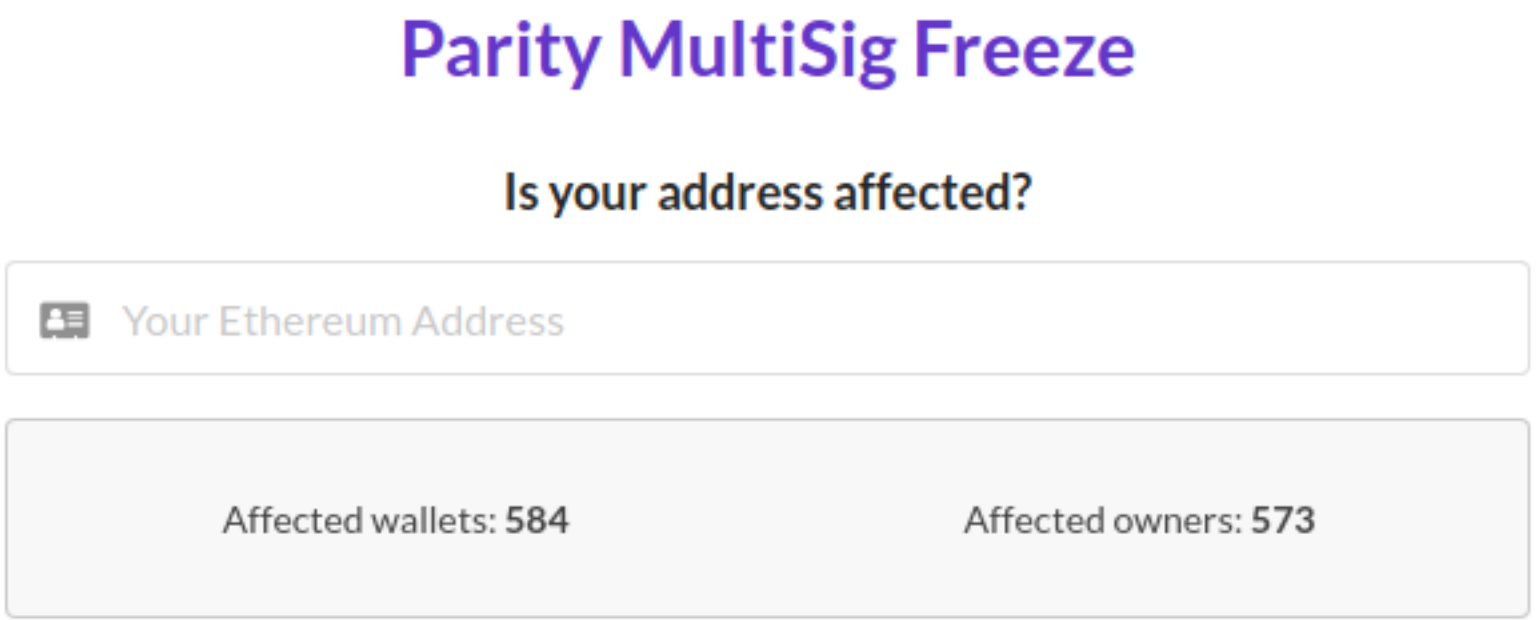
```
contract WalletLibrary {
     address owner;
     // called by constructor
     function initWallet(address _owner) {
         require(initialized == false);
         initialized = true;
         owner = _owner;
         // ... more setup ...
     }
     function changeOwner(address _new_owner)
     function () payable {
         // ... receive money, log events,
...
     }
     function withdraw(uint amount);
}
```

0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

Slide credit to Andrew Miller

After a developer stumbled upon the bug – "accidentally" deleting a code library for the Parity wallet and freezing more than $152 million-worth of ether – several startups and open-source projects that recently launched initial coin offerings (ICOs) have come forward, stating that theirs are among the 151 addresses impacted by the software failure.

> According to crypto eli5, 151 wallets have been frozen, with their balances being 513,743 ETH or $152 million in total. Parity Technologies announce that 573 wallets have been affected and their total balance is unknown.



Slide credit to Andrew Miller

No one knows who devops199 was!

Tienus @Tienus

@devops199 you are the one that called the kill tx?

devops199 @devops199

yes

i'm eth newbie..just learning

qx133 @qx133

you are famous now haha

devops199 @devops199

sending kill() destroy() to random contracts

you can see my history

😟((((((((((((((((((((((((((((

Deleted user

ghost

Slide credit to Andrew Miller

Case 1: lessons learned

- In the world of smart contract, the cost of even a small mistake can be tremendous
- Be careful with fallback function and low level calls
- Remember to do initialization and access control

Case 2: recall the ERC20 interface

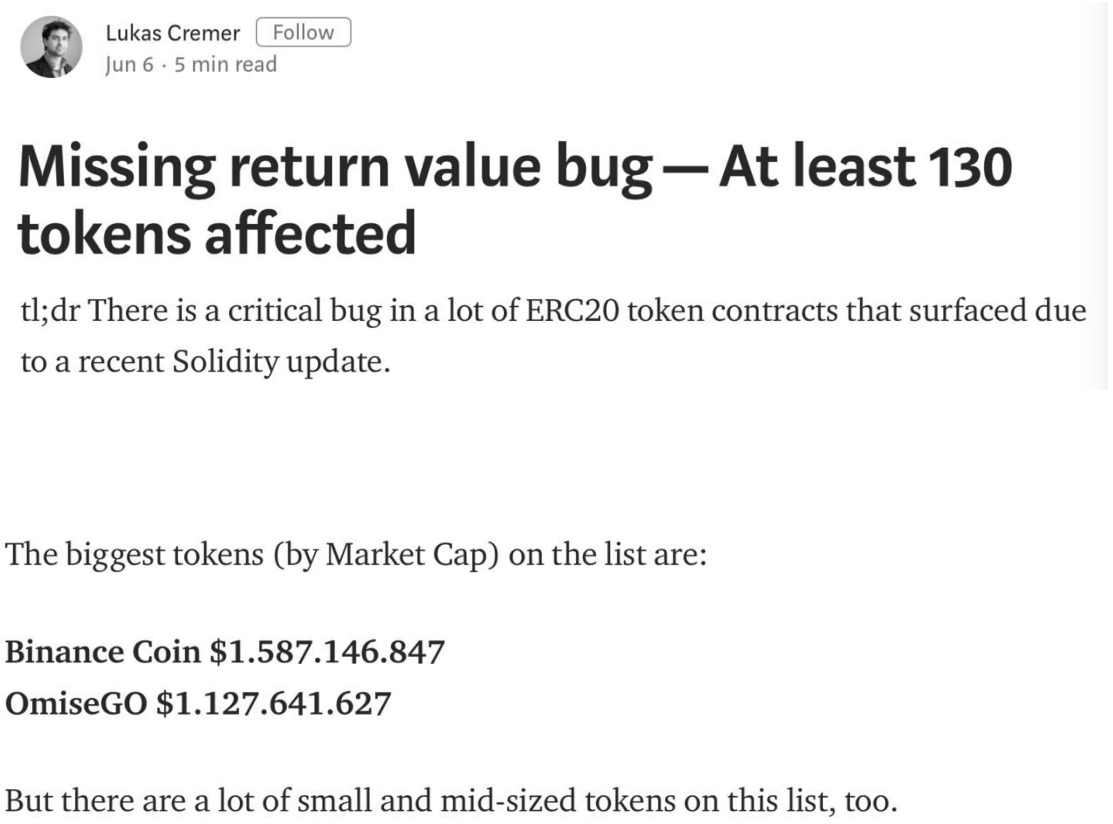
- transfer function

```
function _transfer(address from, address to, uint256 value) internal {
    require(value <= _balances[from]);
    require(to != address(0));

    _balances[from] = _balances[from].sub(value);
    _balances[to] = _balances[to].add(value);
}

function transfer(address to, uint256 value) public returns (bool) {
    _transfer(msg.sender, to, value);
    emit Transfer(msg.sender, to, value);
    return true;
}
```

Case 2: be careful with return value

- Many token contracts didn’t strictly comply to the ERC20 standards
- They implement transfer() without a return value
- If called, the EVM will take whatever found in the memory slot as the return value, presuming the callee is a ERC20 but indeed it is NOT
- Any non-zero value is parsed as true
- Seemly no problem in older compiler <0.4.22
- Compiler since 0.4.22 will check the *RETURNDATASIZE*
  - Revert if shorter than expected
- The transfer() can never be called by a newly compiled contract!
- Missing return is a general problem to all functions

Case 2: impact of missing return



Lukas Cremer Follow
Jun 6 · 5 min read

Missing return value bug — At least 130 tokens affected

tl;dr There is a critical bug in a lot of ERC20 token contracts that surfaced due to a recent Solidity update.

The biggest tokens (by Market Cap) on the list are:

Binance Coin $1.587.146.847
OmiseGO $1.127.641.627

But there are a lot of small and mid-sized tokens on this list, too.

```
Tokens that are not returning a bool in the transfer function.
Only tokens in listed on Etherdelta were checked.

https://raw.githubusercontent.com/etherdelta/etherdelta.github.io/master/config/main.jsor

all these contracts are affected by the missing return value bug. Morte info on this issu
https://github.com/ethereum/solidity/issues/4116



{'addr': '0xae616e72d3d89e847f74e8ace41ca68bbf56af79', 'name': 'GOOD', 'decimals': 6}
{'addr': '0x93e682107d1e9defb0b5ee701c71707a4b2e46bc', 'name': 'MCAP', 'decimals': 8}
{'addr': '0xb97048628db6b661d4c2aa833e95dbe1a905b280', 'name': 'PAY', 'decimals': 18}
{'addr': '0x4470bb87d77b963a013db939be332f927f2b992e', 'name': 'ADX', 'decimals': 4}
{'addr': '0xd26114cd6ee289accf82350c8d8487fedb8a0c07', 'name': 'OMG', 'decimals': 18}
{'addr': '0xb8c77482e45f1f44de1745f52c74426c631bdd52', 'name': 'BNB', 'decimals': 18}
{'addr': '0xf433089366899d83a9f26a773d59ec7ecf30355e', 'name': 'MTL', 'decimals': 8}
{'addr': '0xc63e7b1dece63a77ed7e4aeef5efb3b05c81438d', 'name': 'FUCKOLD', 'decimals': 4}
{'addr': '0xab16e0d25c06cb376259cc18c1de4aca57605589', 'name': 'FUCK', 'decimals': 4}
{'addr': '0xe3818504c1b32bf1557b16c238b2e01fd3149c17', 'name': 'PLR', 'decimals': 18}
{'addr': '0xe2e6d4be086c6938b53b22144855eef674281639', 'name': 'LNK', 'decimals': 18}
{'addr': '0x2bdc0d42996017fce214b21607a515da41a9e0c5', 'name': 'SKIN', 'decimals': 6}
{'addr': '0xea1f346faf023f974eb5adaf088bbcdf02d761f4', 'name': 'TIX', 'decimals': 18}
{'addr': '0x177d39ac676ed1c67a2b268ad7f1e58826e5b0af', 'name': 'CDT', 'decimals': 18}
{'addr': '0x56ba2ee7890461f463f7be02aac3099f6d5811a8', 'name': 'CAT', 'decimals': 18}
{'addr': '0x08fd34559f2ed8585d3810b4d96ab8a05c9f97c5', 'name': 'CLRT', 'decimals': 18}
{'addr': '0x2a05d22db079bc40c2f77a1d1ff703a56e631cc1', 'name': 'BAS', 'decimals': 8}
{'addr': '0xdc0c22285b61405aae01cba2530b6dd5cd328da7', 'name': 'KTN', 'decimals': 6}
{'addr': '0x9e77d5a1251b6f7d456722a6eac6d2d5980bd891', 'name': 'BRAT', 'decimals': 8}
{'addr': '0x202e295df742befa5e94e9123149360db9d9f2dc', 'name': 'NIH', 'decimals': 8}
{'addr': '0xfb12e3cca983b9f59d90912fd17f8d745a8b2953', 'name': 'LUCK', 'decimals': 0}
{'addr': '0x94298f1e0ab2dfad6eeffb1426846a3c29d98090', 'name': 'MyB', 'decimals': 8}
```

Case 2: one reason why so many victims

This issue is clearly a bug in the token contract. The change in Solidity only brought this bug to daylight. One reason why there are so many BadTokens is that at one point OpenZeppelin implemented the wrong interface in their framework. Between 17 March 2017 and 13 July 2017 the interface was wrong:

Tree: 52120a8c42 ▾ openzeppelin-solidity / contracts / token / ERC20Basic.sol Find file Copy path

maraoz solidity version 0.4.8 52b66c7 on Feb 22, 2017

1 contributor

15 lines (12 sloc) | 362 Bytes Raw Blame History

```
1  pragma solidity ^0.4.8;
2
3
4  /*
5   * ERC20Basic
6   * Simpler version of ERC20 interface
7   * see https://github.com/ethereum/EIPs/issues/20
8   */
9  contract ERC20Basic {
10   uint public totalSupply;
11   function balanceOf(address who) constant returns (uint);
12   function transfer(address to, uint value);
13   event Transfer(address indexed from, address indexed to, uint value);
14 }
```

Case 2: lessons learned

- Always adhere to the standards
- The language and compiler iteration is unprecedentedly fast
  - Subtle problem that does not surface will eventually do
- Even the security experts can make sloppy mistake
  - You are the one responsible for your code

Case 3: let's add some real "fancy" function

Batching transfers to save gas cost

Contract Source Code </>

```
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 }
```

BeautyChain (BEC) 0xC5d105E63711398aF9bbff092d4B6769C82F793D

Case 3: let's add some real "fancy" function

batchOverflow bug (CVE-2018–10299)

Contract Source Code </>

```
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 }
```

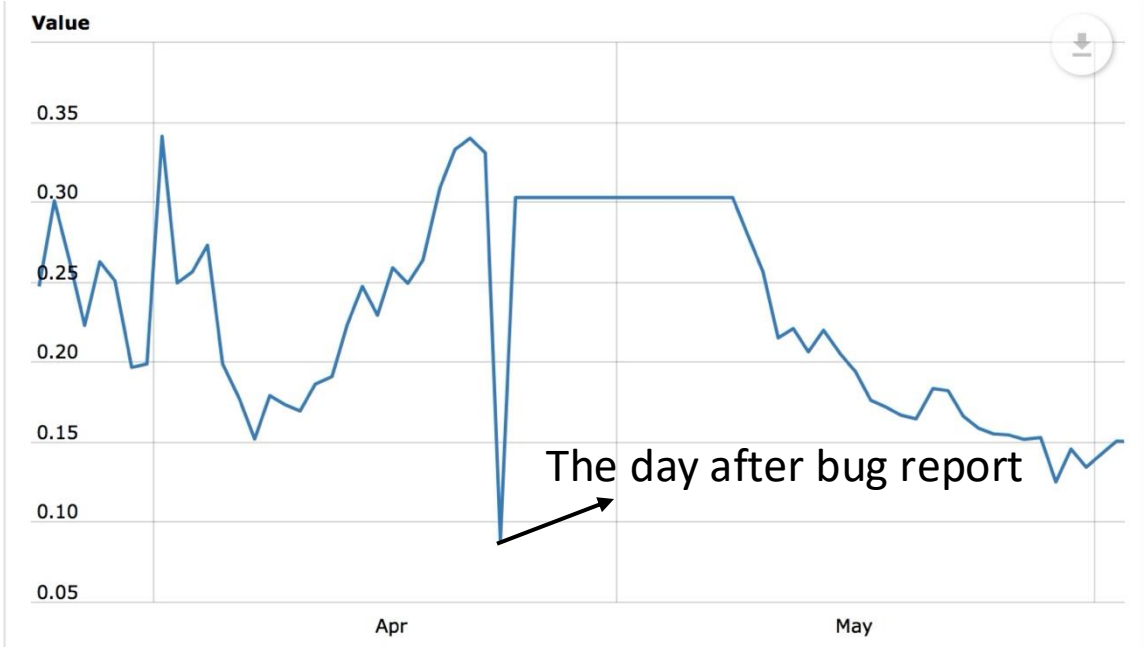
_value=2^255 = 0x800…000 (63 0's), cnt=2

amount = 0

Get 2^255 tokens for free!

BeautyChain (BEC) 0xC5d105E63711398aF9bbff092d4B6769C82F793D

Case 3: impact of batchOverflow



https://www.worldcoinindex.com/coin/bec

ERC-20 Tokens Deposit Suspended

OKEx
April 25, 2018 14:09 · Created

Dear valued customers,

We are suspending the deposits of all ERC-20 tokens due to the discovery of a new smart contract bug - "BatchOverFlow". By exploiting the bug, attackers can generate an extremely large amount of tokens, and deposit them into a normal address. This makes many of the ERC-20 tokens vulnerable to price manipulations of the attackers.

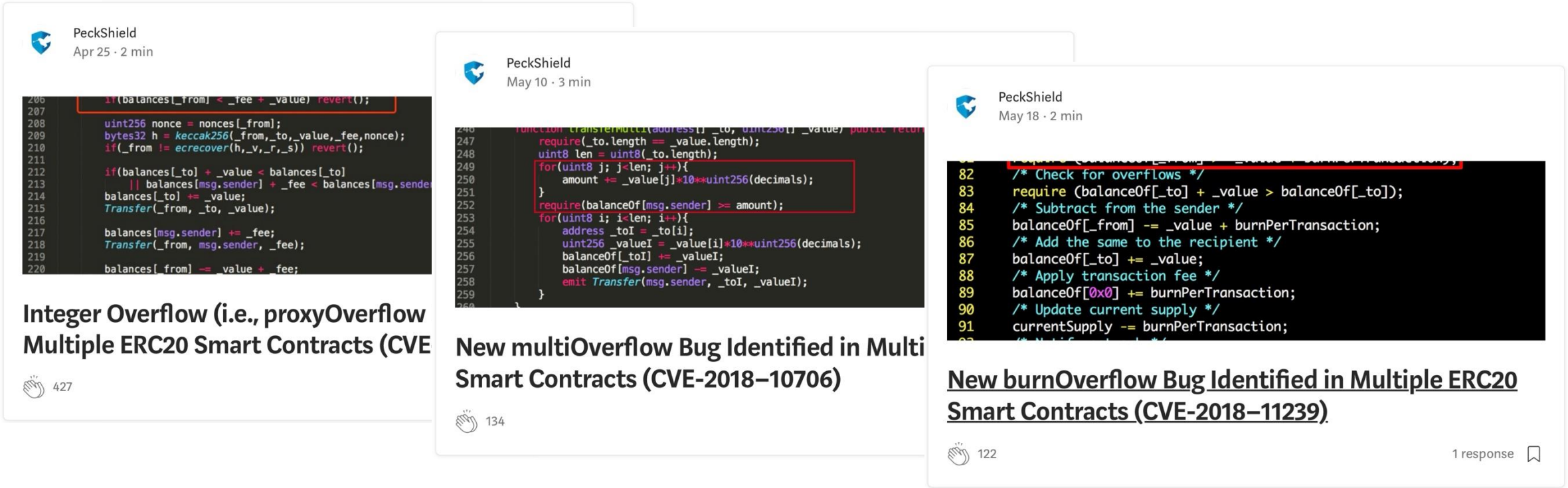
https://support.okex.com/hc/en-us/articles/360003019292-ERC-20-Tokens-Deposit-Suspended

More than a dozen of other contracts are affected

First reported by PeckShield

Case 3: integer overflow bugs are everywhere

- A series of follow-up reports afterwards



A lot more in the wild …

Case 3: simple fix

- Always do safety checking and revert
  - E.g., SafeMath library


```
/**
* @dev Adds two numbers, reverts on overflow.
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  require(c >= a);

  return c;
}
```

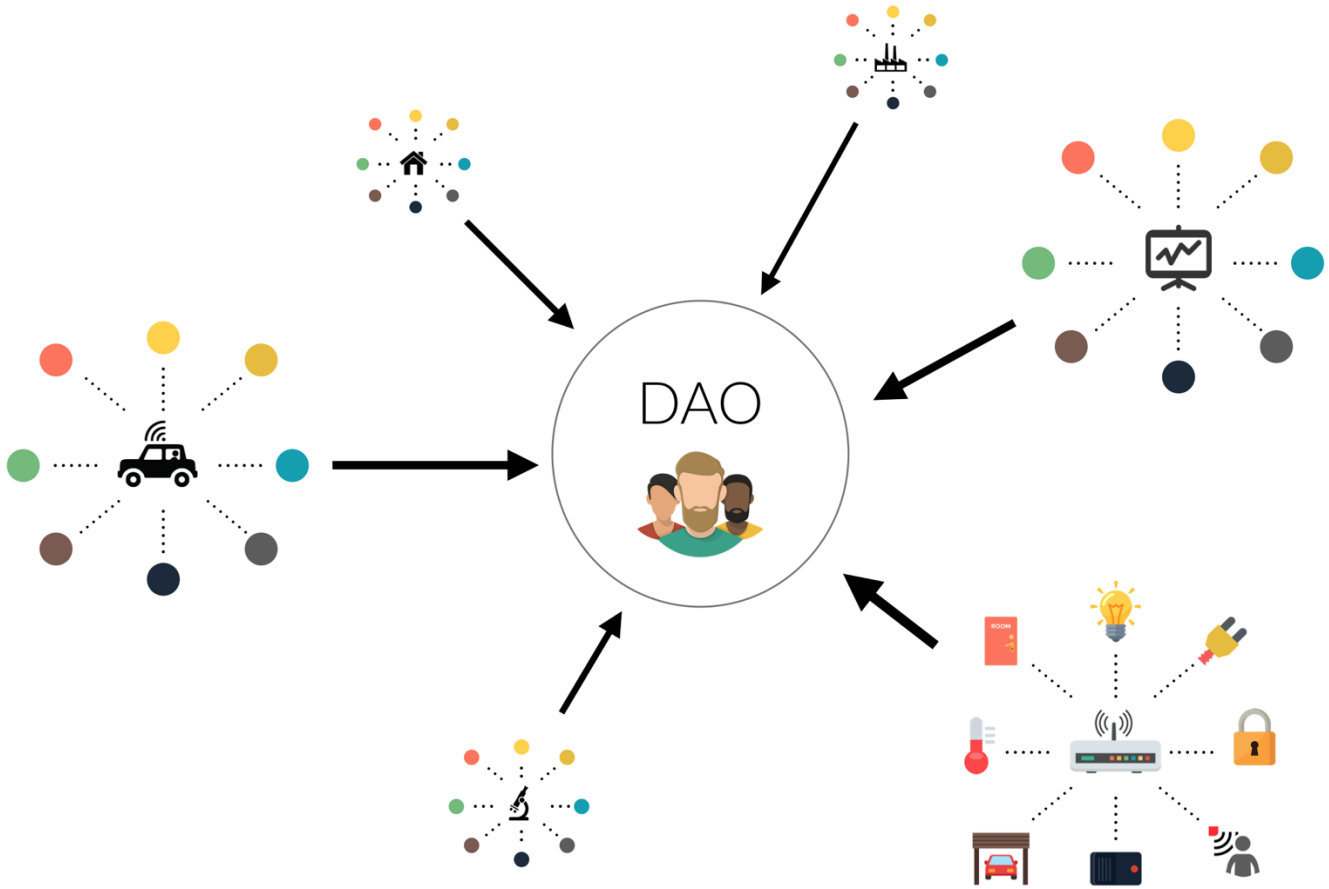
But keep in mind that security experts make mistake too!
Recall the missing value bug in transfer() …

Case 3: lessons learned

- Trivial integer overflow/underflow leads to massive loss
- Remember to use SafeMath throughout the code
- Don’t dumbly copy-and-paste around from the internet
  - You are the one responsible for your code

A side note: the notorious DAO attack

DAO = decentralized autonomous organization



https://media.blockchainhub.net/wp-content/uploads/2016/08/mother.png

Timeline and Aftermath of The DAO Attack

- June 12: slock.it developers announce that a bug is found, but no funds at risk
- June 17 (Morning): attacker drains ⅓ of the DAO's Ether ($50M) over 24 hrs

  Attacker's funds were trapped in a subcontract for ***40 days*** (July 27)
- June 17 (Evening): Eth Foundation proposes a "Soft Fork" to freeze the funds
- June 28: researchers publish a flaw in the Soft Fork Proposal
- July 15 (Morning): Eth Foundation proposes a "Hard Fork" to recover funds
- July 15 (Evening): "Ethereum Classic" manifesto published on github
- July 19: "Hard Fork" moves funds from attacker's contract to recovery contract

  Ethereum Classic blockchain survives and is traded on exchanges

Both Ethereum and Ethereum Classic are both around, reached new peaks

Slide credit to Andrew Miller

Excerpt from DAO contract (simplified)

Contract DAO:

```
mapping (address => int64) balances;

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


Slide credit to Andrew Miller

Re-entrancy hazards in Ethereum

Balance 100

Wallet Contract

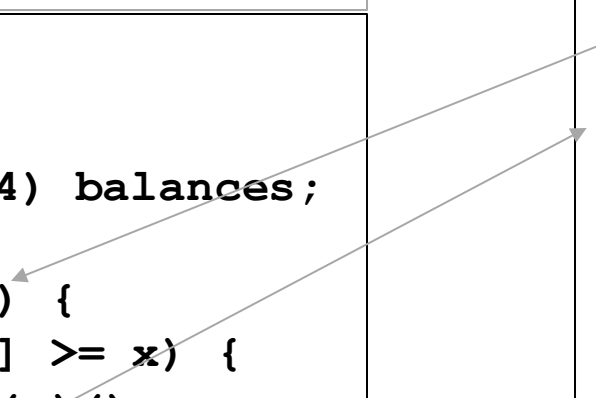
```
function doWithdraw() {
    A.withdraw(100);
}
function() {
    EventMoneyReceived(msg.value);
}
```

Ether Balance 200

balances[user] 0

Contract DAO:

```
mapping (address => int64) balances;

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



Slide credit to Andrew Miller

Re-entrancy hazards in Ethereum

Balance	300

Attacker Contract

```
function startAttack() {
    A.withdraw(100);
}
function() {
    A.withdraw(100);
}
```

Ether Balance	0
balances[attacker]	-300

Contract DAO:

```
mapping (address => int64) balances;

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

Slide credit to Andrew Miller

Where to look for more Ethereum smart contract attacks and vulnerabilities?

Decentralized Application Security Project (DASP) Top 10 of 2018

1. Reentrancy
2. Access Control
3. Arithmetic Issues
4. Unchecked Return Values For Low Level Calls
5. Denial of Service
6. Bad Randomness
7. Front-Running
8. Time manipulation
9. Short Address Attack
10. Unknown Unknowns

https://www.dasp.co

Awesome Buggy ERC20 Tokens

A. Implementation Bugs	A1. batchTransfer-overflow
	A2. totalsupply-overflow
	A3. verify-invalid-by-overflow
	A4. owner-control-sell-price-for-overflow
	A5. owner-overweight-token-by-overflow
	A6. owner-decrease-balance-by-mint-by-overflow
	A7. excess-allocation-by-overflow
	A8. excess-mint-token-by-overflow
	A9. excess-buy-token-by-overflow
	A10. verify-reverse-in-transferFrom
	A11. pauseTransfer-anyone
	A12. transferProxy-keccak256
	A13. approveProxy-keccak256
	A14. constructor-case-insentive
	A15. custom-fallback-bypass-ds-auth
	A16. custom-call-abuse
	A17. setowner-anyone
	A18. allowAnyone
	A19. approve-with-balance-verify
	A20. re-approve
B. Nonstandard Functions	B1. transfer-no-return
	B2. approve-no-return
	B3. transferFrom-no-return
	B4. no-decimals
	B5. no-name
	B6. no-symbol
	B7. no-Approval
C. Excessive Authorities	C1. centralAccount-transfer-anyone

https://github.com/sec-bit/awesome-buggy-erc20-tokens

Where to look for more Ethereum security best practices?

- Online resources
  - “Ethereum Smart Contract Security Best Practices” from ConsenSys
  - OpenZeppelin library and blogs

- Academic papers
  - “Making Smart Contracts Smarter”, in CCS’16
  - “Step by Step Towards Creating a Safe Smart Contract- Lessons and Insights from a Cryptocurrency Lab”, in FC’18

Related References

- Konstantinos Christidis and Michael Devetsikiotis. “Blockchains and Smart Contracts for the Internet of Things”, IEEE Access, 2016.
- The ZILLIQA Technical Whitepaper. Online at https://docs.zilliqa.com/whitepaper.pdf
- The Ethereum White Paper. Online at https://github.com/ethereum/wiki/wiki/White-Paper
- The Solidity Programming Language for Ethereum. Online at https://solidity.readthedocs.io/en/v0.5.4/
- The Ethereum Yellow Paper. Online at https://ethereum.github.io/yellowpaper/paper.pdf
- ERC20 Token Standard. Online at https://theethereum.wiki/w/index.php/ERC20_Token_Standard
- Adam Back, Matt Corallo, et al. “Enabling Blockchain Innovations with Pegged Sidechains”, 2014

Ahmed E. Kosba et al. “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts”, in Proc. of IEEE S&P’ 16.

Ari Juels et al. “The Ring of Gyges: Investigating the Future of Criminal Smart Contracts”, in Proc. of ACM CCS’16.

Florian Tram`er et al. “Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge”, in Proc. of IEEE Euro S&P’ 17.

Lorenz Breidenbach et al. “Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts”, in Proc. of Usenix Security’ 18

Storj project, online at https://storj.io/whitepaper/