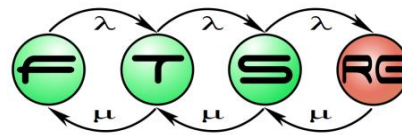


Not So Smart Contracts Vulnerabilities and Verification

Blockchain Technologies and Applications

Ákos Hajdu, hajdua@mit.bme.hu

Imre Kocsis, ikocsis@mit.bme.hu



Motivating example

■ Simplified version of the DAO hack

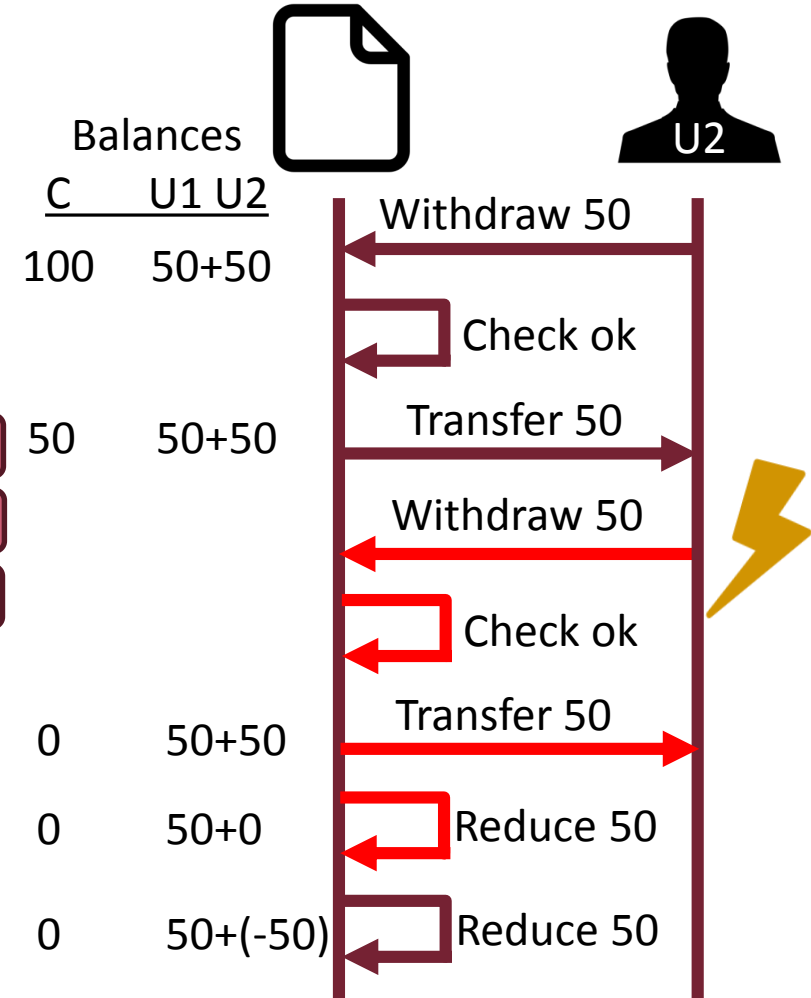
```

contract Bank {
  mapping(address=>uint) balances;
  function deposit() payable {
    balances[msg.sender] += msg.value;
  }
  function withdraw(uint amount) {
    require (balances[msg.sender] >= amount);
    if (!msg.sender.call.value(amount)("")) {
      revert();
    }
    balances[msg.sender] -= amount;
  }
}

```

Data: balance of each user
 Operation: deposit money
 Receive money
 Increase balance
 Operation: withdraw money
 Check balance
 Transfer money
 Reduce balance

Attack scenario example



Vikram Dhillon, David Metcalfe, and Max Hooper. *The DAO hacked*.
 In: *Blockchain Enabled Applications*, pp. 67–78. Springer, 2017.

More motivating examples

A Hacking of More Than \$50 Million Dashes Hopes in the World of Virtual Currency

By Nathaniel Popper

June 17, 2016

A hacker on Friday siphoned

ETHEREUM, TECHNOLOGY

BatchOverflow Ethereum Token Deposits



Sam Town



April 25, 2018



3 min read



5827 Views

Someone 'Accidentally' Locked Away \$150M Worth of Other People's Ethereum Funds

And a hard fork is on the table.

Parity Multisig Hacked. Again

National Vulnerability Database (NVD)
400+ vulnerability records for blockchain
95%+ are programming errors in contracts

of the [ANY Parity] multi-

panies/ICOs are using Parity-generated multisig wallets.

About \$300M is frozen and (probably) lost forever.

<https://nvd.nist.gov/vuln/>

Where do the problems come from?

- New paradigm for developers
 - *Cf. sequential vs. parallel programming*
 - Accounts, blockchain, transactions, mining, ...
 - Semantic misalignments
 - Easy to make errors
- Problems at different levels
 - Programming language / contracts
 - Execution engine
 - Blockchain and cross-peer protocols



Atzei, Bartoletti, Cimoli - A survey of attacks on Ethereum smart contracts (2017)

Luu, Chu, Olickel, Saxena, Hobor - Making Smart Contracts Smarter (2016)

Nikolic, Kolluri, Sergey, Saxena, Hobor - Finding The Greedy, Prodigal, and Suicidal Contracts at Scale (2018)

https://consensys.github.io/smart-contract-best-practices/known_attacks/


<https://solidity.readthedocs.io/en/latest/security-considerations.html>

What can possibly go wrong?


■ Programming language / contracts

- Call to the unknown
- Gasless send
- Mishandled exceptions
- Type casts
- Reentrancy
- Keeping secrets
- Unchecked caller
- Input validation


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



```
function withdraw(uint amount) {  
    require(balances[msg.sender] >= amount);  
    msg.sender.send(amount);  
    balances[msg.sender] -= amount;  
}
```



```
function withdraw(uint amount) {  
    require(balances[msg.sender] >= amount);  
    if (!msg.sender.send(amount)) revert();  
    balances[msg.sender] -= amount;  
}
```



What can possibly go wrong?

- Execution engine
 - Under/overflows
 - Immutable bugs
 - Ether lost in transfer
 - Stack size limit

```
uint8 x = 255;  
uint8 y = 1;  
  
uint8 z = x + y; // z == 0
```

```
int8 x = 127;  
int8 y = 1;  
  
int8 z = x + y; // z == -128
```

The BECToken

```
contract BecTokenSimplified {  
    using SafeMath for uint256;
```

Total tokens

```
    uint256 public totalSupply;  
    mapping(address => uint256) balances;
```

Balance of each user

```
    constructor() {  
        totalSupply = 7000000000 * (10**18);  
        balances[msg.sender] = totalSupply;  
    }
```

Initialization

Creator gets 7×10^{27}

```
    function batchTransfer(address[] receivers, uint256 value) returns (bool) {  
        uint256 amount = receivers.length * value;  
        require(value > 0 && balances[msg.sender] >= amount);  
        balances[msg.sender] = balances[msg.sender].sub(amount);  
        for (uint i = 0; i < receivers.length; i++) {  
            balances[receivers[i]] = balances[receivers[i]].add(value);  
        }  
        return true;  
    }  
}
```

Batch transfer value to N receivers

Total amount = value x N

Reduce sender balance by total

Increase receiver balances
by value N times

The BECToken

Creator	Attacker1	Attacker2	Attacker3	Attacker4	Attacker5
7×10^{27}	0	0	0	0	0

$$\sum 7 \times 10^{27}$$

- Let's „print” money

value = 28948022309329048855892746252171976963317496166410141009864396001978282409984;
attacker1: bectoken.batchTransfer([attacker2, attacker3, attacker4, attacker5], value)

N = 4

amount = value * N = 0



Creator	Attacker1	Attacker2	Attacker3	Attacker4	Attacker5
7×10^{27}	0	2.9×10^{76}	2.9×10^{76}	2.9×10^{76}	2.9×10^{76}

$$\sum 1.16 \times 10^{77}$$

- Really happened (with different parameters)

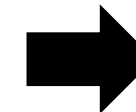
TxHash	Age	From		To	Quantity
0xad89ff16fd1ebe3...	14 hrs 7 mins ago	0x09a34e01fbaa49f...	IN	0x0e823ffe0187275...	57,896,044,618,658,100,000,000,000...

<https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536>

What can possibly go wrong?

- Blockchain and cross-peer protocols
 - Unpredictable state
 - Transaction ordering dependency
 - Generating randomness
 - Time constraints
 - Timestamp dependency

```
contract Market {  
    uint public price;  
    uint public stock;  
    ...  
    function setPrice(uint _price) {  
        if (msg.sender == owner)  
            price = _price;  
    }  
    function buy(uint quantity) {  
        if (msg.value < quantity * price ||  
            stock < quantity) revert();  
        stock -= quantity;  
        ...  
    }  
}
```



Why is this important?

- Real consequences
 - Contracts manage real-life assets
 - Ethereum: 22B USD market cap
 - Not only financial aspects
 - E.g., smart lock
- Permanent
 - Once deployed, no patching¹
 - No transaction reverting²
 - Compile time verification needed
- Public platforms: open world
 - Available to everyone
 - Everyone sees the code
 - Everyone can send transactions

Let's do
verification!

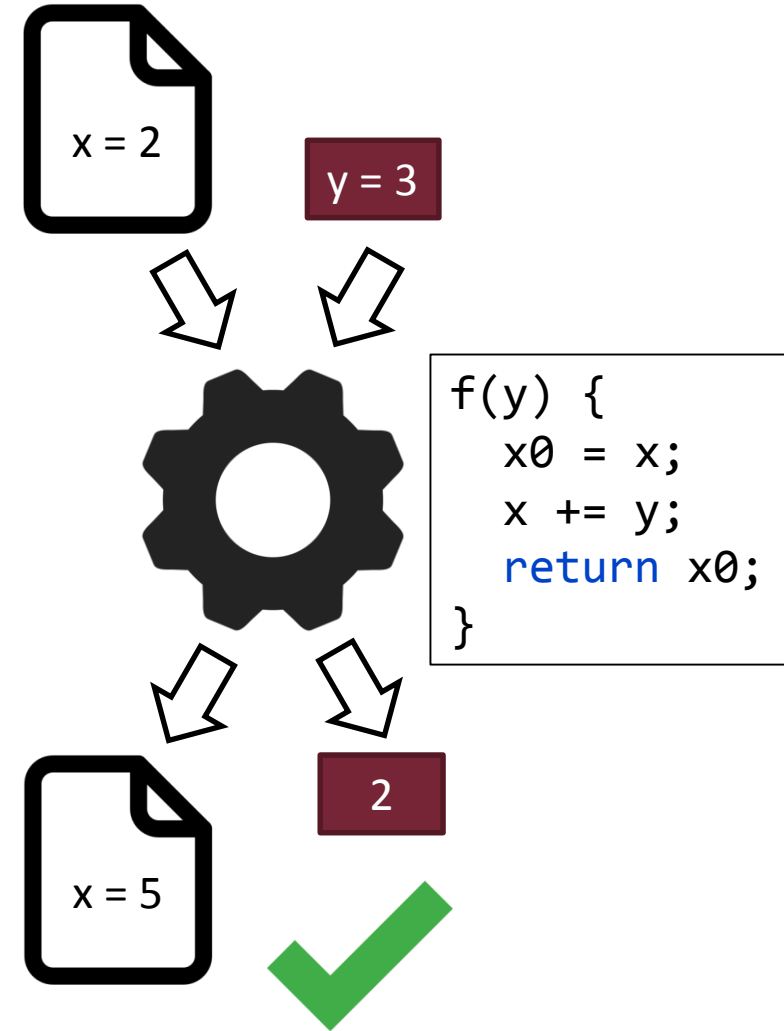
¹There are patterns to kill a contract or redirect calls, but that brings up new vulnerabilities

²Apart from solutions involving a central authority

VERIFICATION APPROACHES

Testing

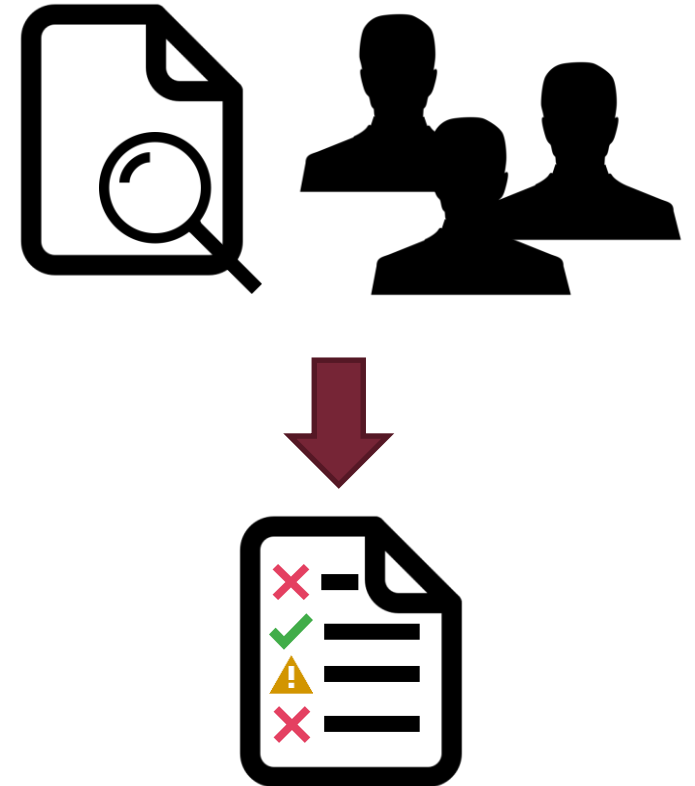
- Contract state + input → expected state + output
 - Traditional testing strategies and techniques
- Frameworks help (e.g., Truffle)
 - Setup test network with initial state
 - Execute steps, check state and output
- Advantages and drawbacks
 - Efficient in finding bugs, understanding the code
 - Test high-level business logic
 - Manual process
 - Cannot test every state and input
 - Complex scenarios: other users, contracts, miners
 - DAO requires an attacker contract



<https://truffleframework.com/docs/truffle/testing/testing-your-contracts>
<https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html>

Audit / Review

- Experts review and analyze the contracts
 - Contact, get a quote
 - Perform audit
 - Report
 - Fix issues
- Advantages and drawbacks
 - Detailed, high/low-level analysis
 - Expensive
 - Time consuming, non-interactive
 - Experts are human too, can make mistakes

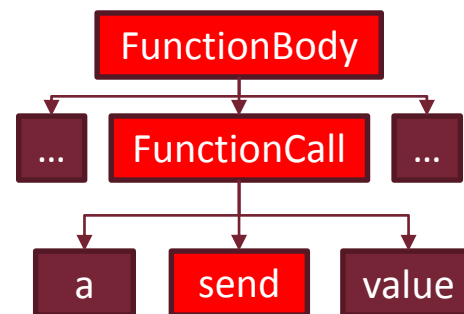
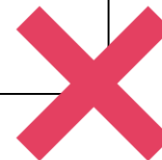


<https://zeppelin.solutions/security>
<https://solidity.readthedocs.io/en/v0.5.4/security-considerations.html>

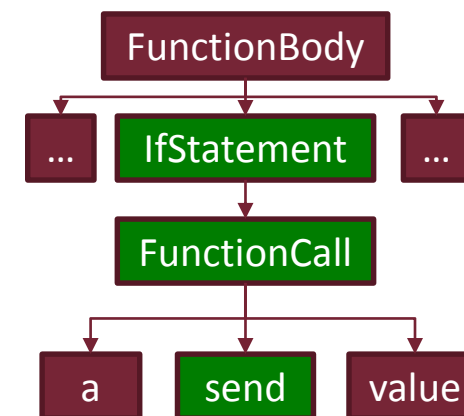
Vulnerability patterns

- Pattern matching
 - Abstract syntax tree (AST)
 - Violation/compliance patterns
- Advantages and drawbacks
 - Fully automated
 - Scalable to large contracts
 - False alarms
 - Missed bugs
 - No high-level properties

```
function f() {  
  ...  
  a.send(value);  
  ...  
}
```



```
function g() {  
  ...  
  if (a.send(value)) ...;  
  ...  
}
```



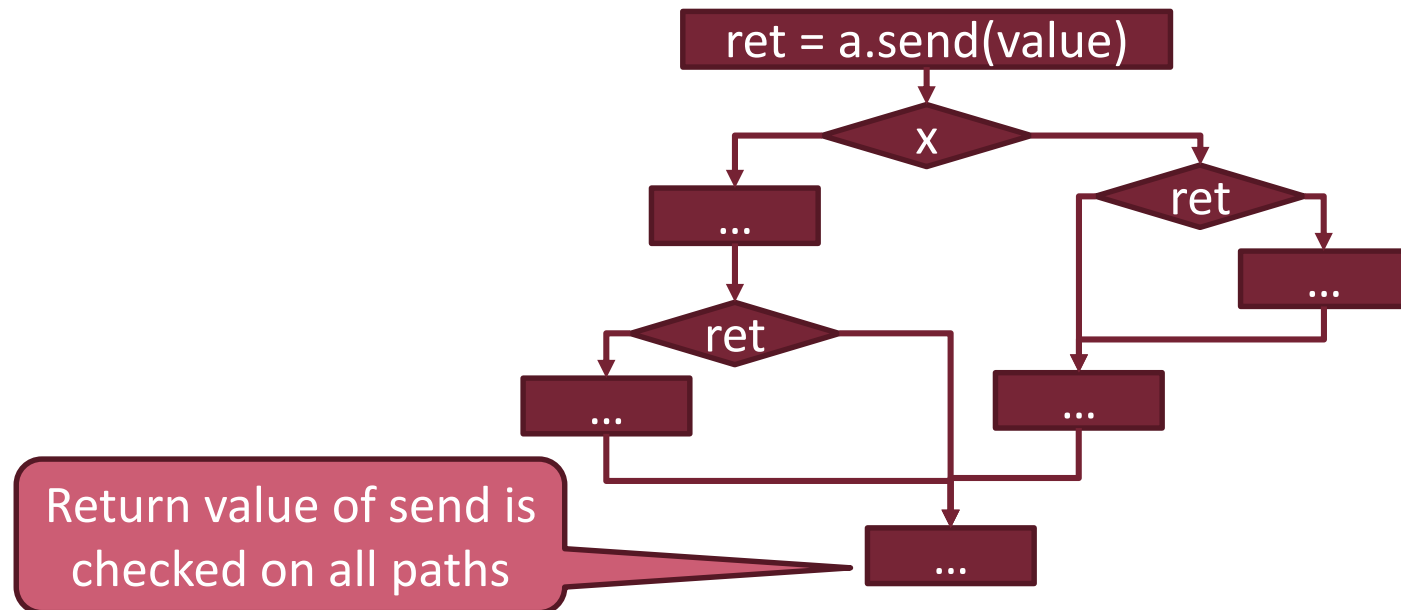
Luu, Chu, Olickel, Saxena, Hobor - Making Smart Contracts Smarter (2016)

Tsankov, Dan, Drachler-Cohen, Gervais, Bunzli, Vechev - Securify Practical Security Analysis of Smart Contracts (2018)

Symbolic execution

- Reason about paths symbolically
 - Control flow patterns
 - Data flow patterns
- Advantages and drawbacks
 - Similar to pattern-based
 - Higher-level patterns
 - Less false alarms
 - Less scalable


```
ret = a.send(value);  
if (x) {  
  if (ret) { ... }  
  ...  
} else {  
  ...  
  if (ret) { ... }  
}  
...
```



Formal verification techniques

- Translate code to formal representation
 - Apply mathematical reasoning
 - Formal requirement needed too
 - E.g., assert, require, annotations
- Advantages and drawbacks
 - Automated
 - High-level properties
 - Fully formal, real errors, bugs not missed
 - Depending on assumptions and abstractions
 - Might suffer from scalability issues
 - Extra developer effort for requirements

```
function abs(int x) returns (int) {  
  int y;  
  if (x >= 0) y = x;  
  else y = -x;  
  assert(y >= 0);  
  return y;  
}
```



$$\begin{array}{ccc} (x \geq 0 \wedge y = x) & ? & \\ \vee & \Rightarrow & y \geq 0 \\ (x \not\geq 0 \wedge y = -x) & & \end{array}$$

<https://github.com/SRI-CSL/solidity>

D'Silva, Kroening, Weissenbacher – A Survey of Automated Techniques for Formal Software Verification (2008)

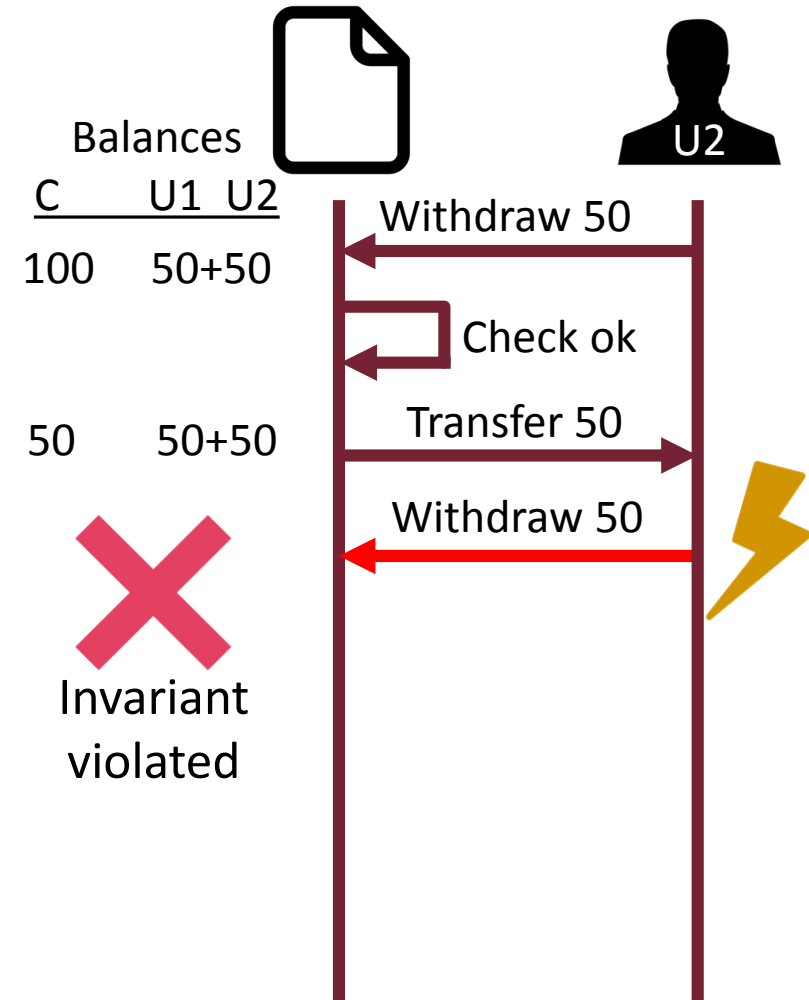
Reentrancy revisited

Invariant: must hold before and after every public function call

```
/** @notice invariant this.balance == sum(balances) */
contract Bank {
    mapping(address=>uint) balances;

    function withdraw(uint amount) {
        require (balances[msg.sender] >= amount);
        if (!msg.sender.call.value(amount)("")) {
            revert();
        }
        balances[msg.sender] -= amount;
    }
}
```

Attack scenario example



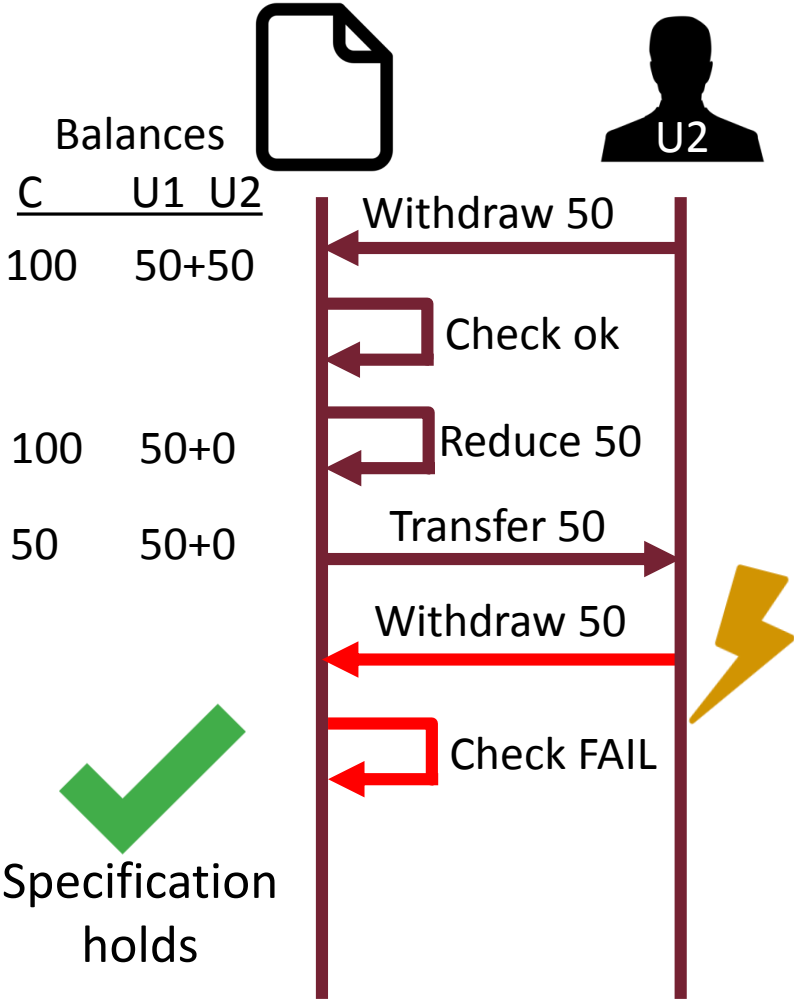
Reentrancy revisited

Invariant: must hold before and after every public function call

```
/** @notice invariant this.balance == sum(balances) */  
contract Bank {  
  
    mapping(address=>uint) balances;  
  
    function withdraw(uint amount) {  
        require (balances[msg.sender] >= amount);  
        balances[msg.sender] -= amount;  
        if (!msg.sender.call.value(amount)("")) {  
            revert();  
        }  
    }  
}
```

First reduce then transfer

Attack scenario example



TOOLS

Tools

- Truffle Suite
 - Development environment and testing framework
- Securify, MythX, Slither
 - Pattern-based, symbolic execution
- Solc-verify, VerX
 - Automated formal verification
- VeriSolid
 - Model-based design and code generation



TRUFFLE



<https://truffleframework.com/>
<https://securify.chainsecurity.com/>
<https://mythx.io/>

<https://github.com/crytic/slither>
<https://github.com/SRI-CSL/solidity>
<https://verx.ch/>
<https://github.com/VeriSolid/smart-contracts>

CONCLUSIONS

Conclusions

- Smart contracts are **not so smart**
 - Infamous hacks: DAO, BECToken
 - Vulnerabilities on different levels
 - Importance of verification
- **Verification** approaches
 - Audit, testing, pattern-based, symbolic execution, formal methods
- Tools
- *For more information, check the links on the slides*

