Automated Verification of Solidity Smart Contracts

Akos Hajdu, Dejan Jovanovic

Presented at the 3rd Winter School in Computer Science and Engineering on Blockchains and Cryptocurrencies

19/12/2018

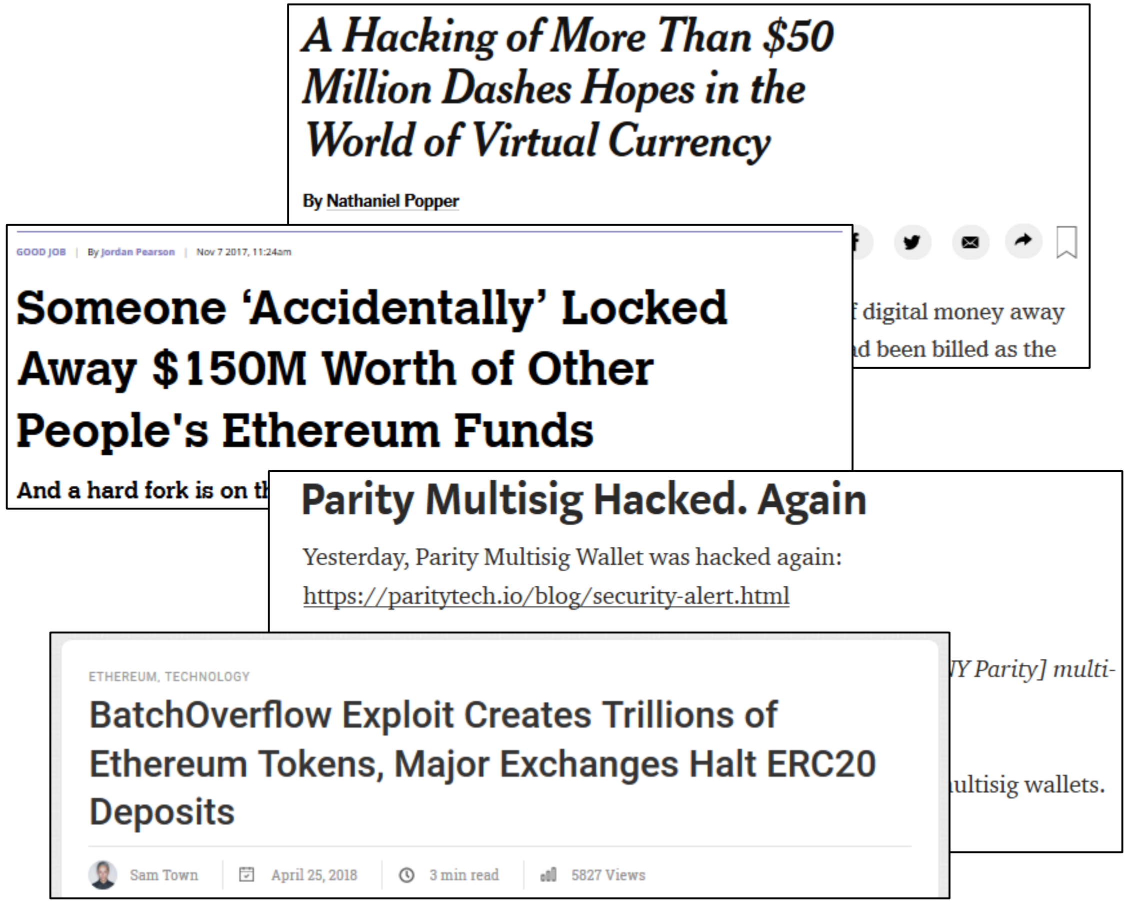
SRI International®

Bugs in smart contracts

- Example: reentrancy in the DAO

```
contract DAO {
  mapping(address=>uint) user_balances;

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



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

By Nathaniel Popper

f digital money away
d been billed as the

GOOD JOB | By Jordan Pearson | Nov 7 2017, 11:24am

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

And a hard fork is on th

Parity Multisig Hacked. Again

Yesterday, Parity Multisig Wallet was hacked again:
https://paritytech.io/blog/security-alert.html

Y Parity] multi-

ultisig wallets.

ETHEREUM, TECHNOLOGY

BatchOverflow Exploit Creates Trillions of Ethereum Tokens, Major Exchanges Halt ERC20 Deposits

Sam Town | April 25, 2018 | 3 min read | 5827 Views

SRI International®

Verification

- We want it to be

Expressive
Wide range of specifications to be expressed

User friendly
Formal methods expertise not required

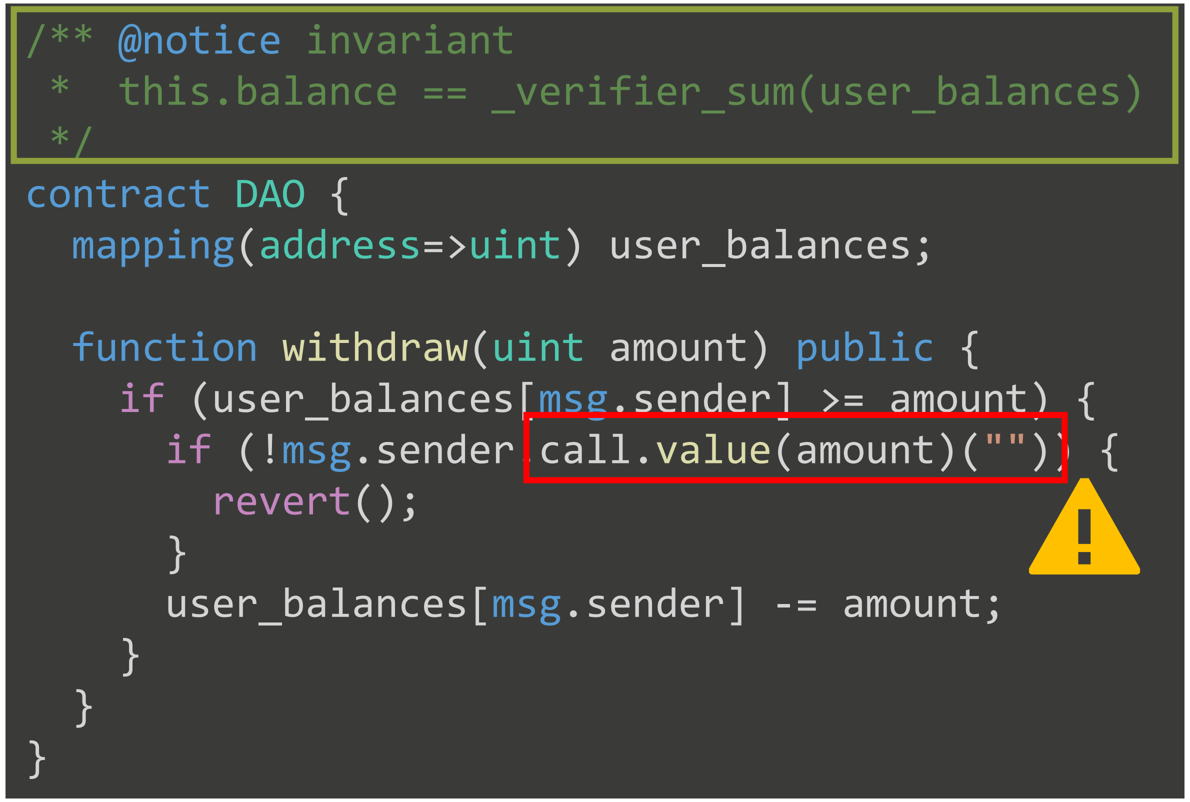
Sound, precise
Find bugs without overwhelming false alarms

Automated
No user interaction required

- Our approach
  - Specification annotations
  - Transformation to intermediate verification language
  - Modular verification using SMT solvers

Specification annotations

- Written in Solidity
  - Contract invariants
  - Function pre- and postconditions
  - Assertions, requires
  - Loop invariants
- Extensions (e.g. sum)
- Overflows



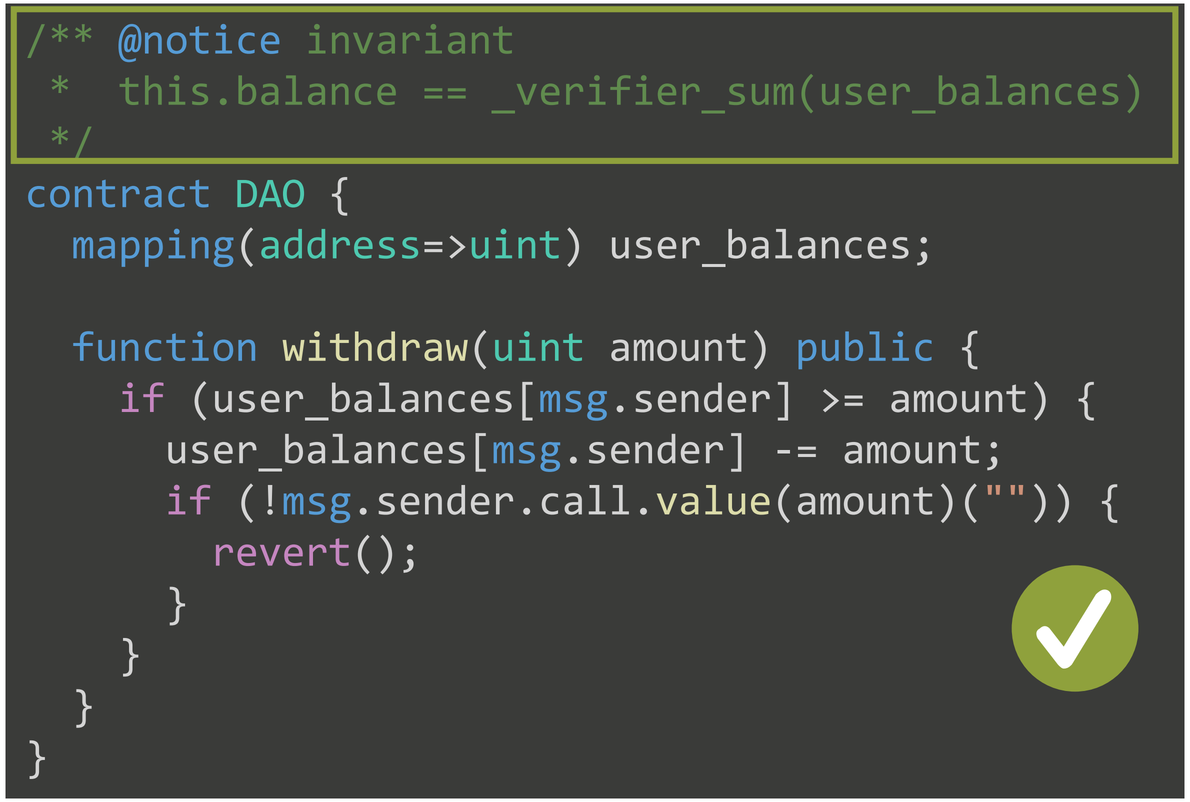
```
/** @notice invariant
 *  this.balance == _verifier_sum(user_balances)
 */
contract DAO {
  mapping(address=>uint) user_balances;

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

SRI International®

Specification annotations

- Written in Solidity
  - Contract invariants
  - Function pre- and postconditions
  - Assertions, requires
  - Loop invariants
- Extensions (e.g. sum)
- Overflows



```
/** @notice invariant
 *  this.balance == _verifier_sum(user_balances)
 */
contract DAO {
  mapping(address=>uint) user_balances;

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

Transformation

- Boogie Intermediate Verification Language
- Basics
  - State variable → global variable
    - Heap-based memory model
  - Function → procedure
- Different arithmetic encodings
  - Scalable bit-precise reasoning
  - Detecting overflows

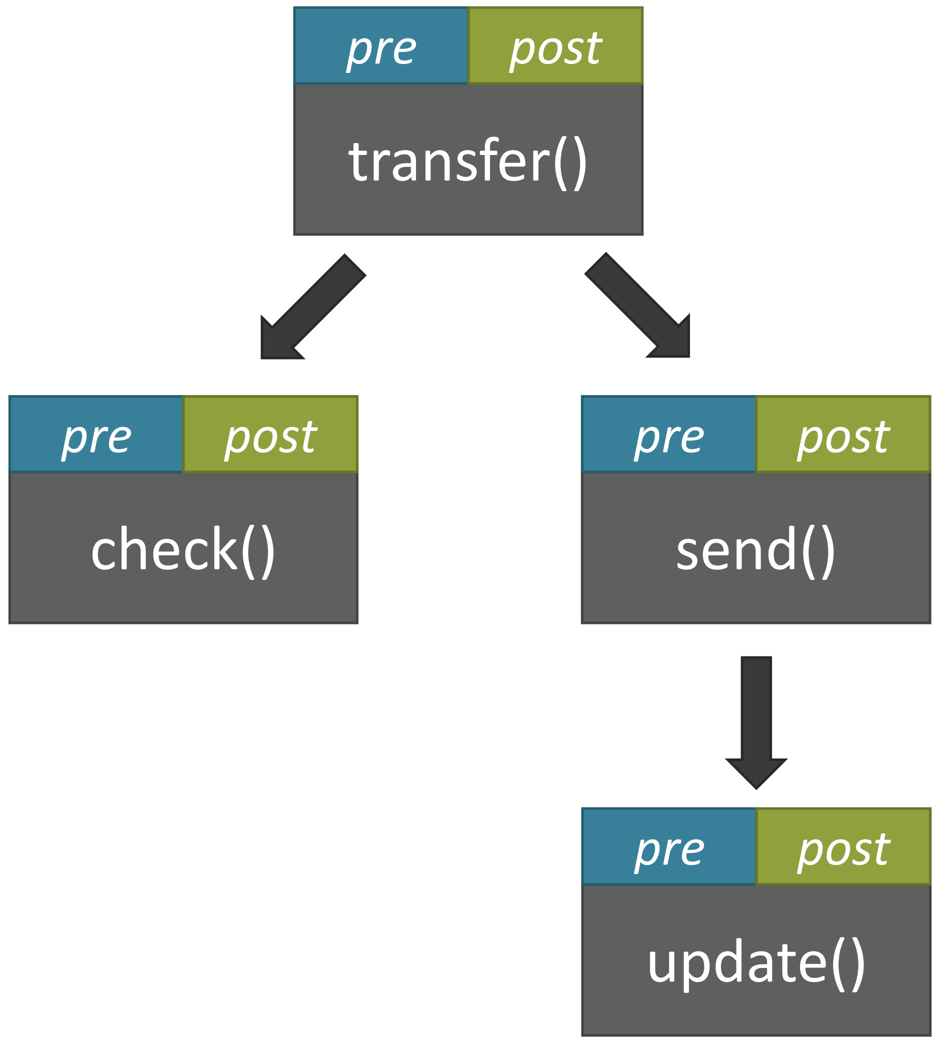
```
var x : int;
var y : int;

procedure add(n : int)
  requires n >= 0;
  requires x == y;
  ensures  x == y;
{
  x := x + n;
  while (y < x)
    invariant y <= x;
  {
    y := y + 1;
  }
}
```

Verification

- Modular verification: units are functions
  - *pre* ∧ *body* → *post*
  - SMT solvers
  - Replace calls with specifications

- Functional correctness with respect to completed transactions
  - Not concerned with termination
  - Not concerned with expected failures



Summary

- Automated formal verification of Solidity smart contracts
  - Specification annotations
  - Transformation to intermediate verification language
  - Modular verification using SMT solvers

Expressive User friendly Sound, precise Automated

inf.mit.bme.hu/en/members/hajdua

csl.sri.com/users/dejan

SRI International®