ICE center@ ETH

CHAINSECURITY

Securify: Practical Security Analysis of Smart Contracts

https://securify.ch

Dr. Petar Tsankov

Scientific Researcher, ICE center, ETH Zurich

Co-founder and Chief Scientist, ChainSecurity AG

⍰ http://www.ptsankov.com/

⍰ @ptsankov

ICE center@ETH

http://ice.ethz.ch

Inter-disciplinary and **inter-department** research center at ETH Zurich

Prof. Martin
Vechev

Prof. Laurent
Vanbever

Dr. Petar
Tsankov

Dr. Dana
Drachsler

Timon
Gehr

Ahmed
El-Hassany

Maria
Apostolaki

Rüdiger
Birkner

Samuel
Steffan

Roland
Meier

Johannes
Kapfhammer

Research @ ICE

Programmable networks

Blockchain security

Safe and interpretable AI

Security and privacy

Research @ ICE

Programmable networks

Blockchain security

Safe and interpretable AI

Security and privacy

What is a Smart Contract?

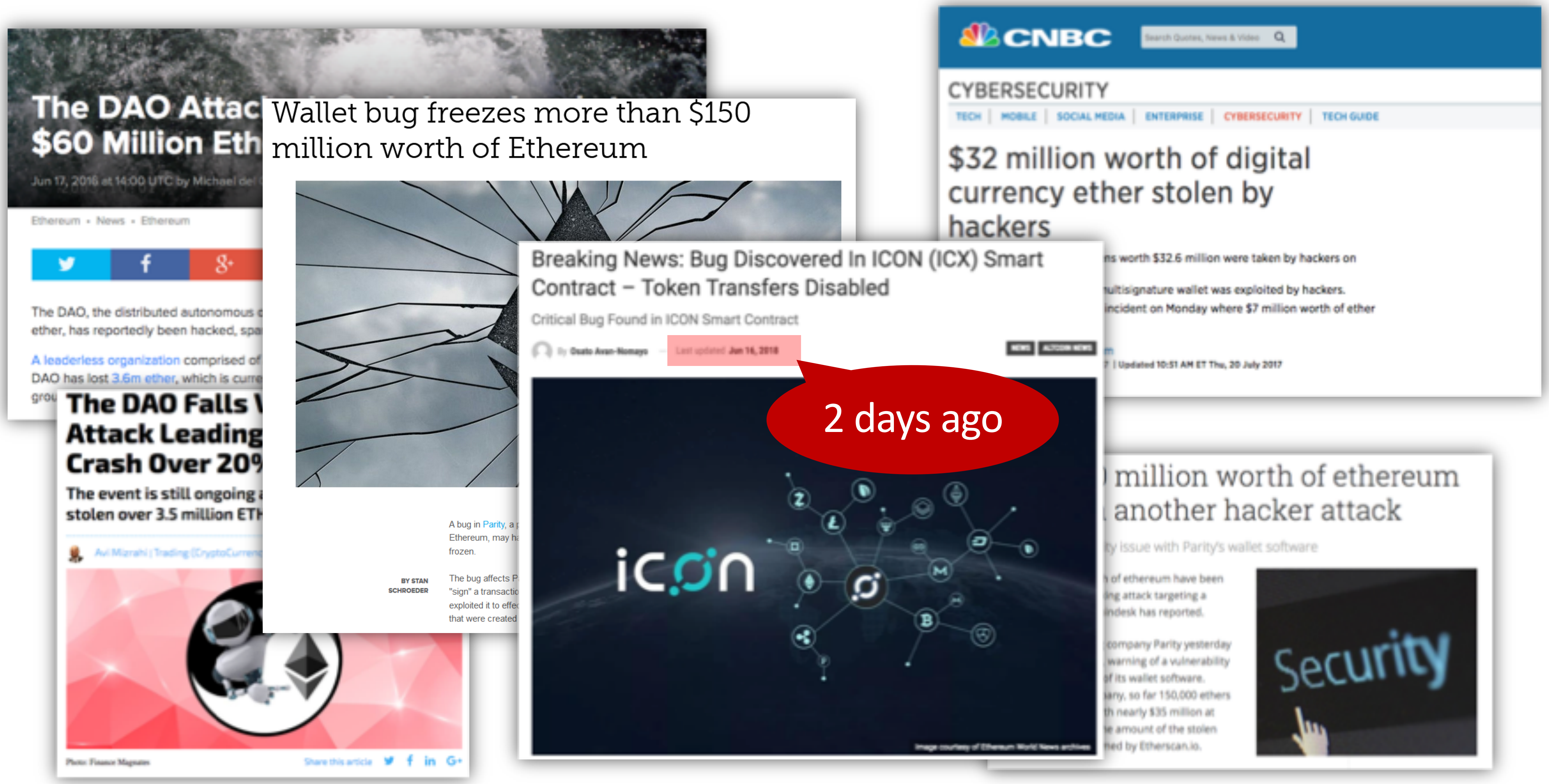
```
mapping(address => uint) balances;

function withdraw() {
  uint amount = balances[msg.sender];
  msg.sender.call.value(amount);
  balances[msg.sender] = 0;
}
```

Transfer ETH to the caller

- Small programs that handle cryptocurrencies
- Written in high-level languages (e.g., Solidity, Vyper)
- Executed on the blockchain (e.g. Ethereum)
- Usually no patching after release

What can happen when programs handle billions worth of USD?

Smart Contract **Security Bugs** in the News



The DAO Attac
$60 Million Eth

Jun 17, 2016 at 14:00 UTC by Michael del

Ethereum • News • Ethereum

The DAO, the distributed autonomous
ether, has reportedly been hacked, spa

A leaderless organization comprised of
DAO has lost 3.6m ether, which is curre
grou

The DAO Falls
Attack Leading
Crash Over 20%

The event is still ongoing
stolen over 3.5 million ETH

Avi Mizrahi | Trading (CryptoCurrenc

Photo: Finance Magnates

Share this article

Wallet bug freezes more than $150 million worth of Ethereum

BY STAN SCHROEDER

A bug in Parity, a
Ethereum, may ha
frozen.

The bug affects P
"sign" a transactio
exploited it to effe
that were created

CNBC

CYBERSECURITY

TECH | MOBILE | SOCIAL MEDIA | ENTERPRISE | CYBERSECURITY | TECH GUIDE

$32 million worth of digital currency ether stolen by hackers

ns worth $32.6 million were taken by hackers on

ultisignature wallet was exploited by hackers.
incident on Monday where $7 million worth of ether

| Updated 10:51 AM ET Thu, 20 July 2017

Breaking News: Bug Discovered In ICON (ICX) Smart Contract – Token Transfers Disabled

Critical Bug Found in ICON Smart Contract

By Osato Avan-Nomayo — Last updated Jun 16, 2018

NEWS | ALTCOIN NEWS

Image courtesy of Ethereum World News archives

2 days ago

million worth of ethereum
another hacker attack

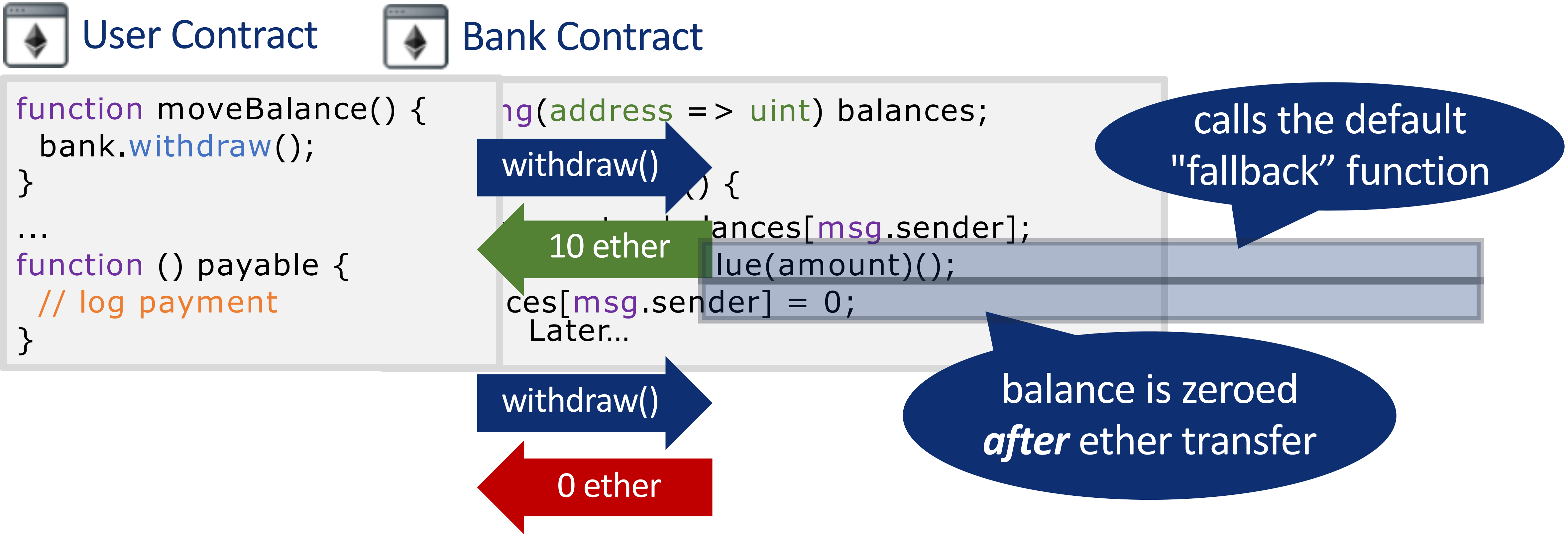
y issue with Parity's wallet software

of ethereum have been
ing attack targeting a
indesk has reported.

company Parity yesterday
warning of a vulnerability
f its wallet software.
any, so far 150,000 ethers
h nearly $35 million at
e amount of the stolen
ned by Etherscan.io.

June 2016: The DAO hack

The DAO hack : Reentrancy



User Contract

Bank Contract

```
function moveBalance() {
  bank.withdraw();
}
...
function () payable {
  // log payment
}
```

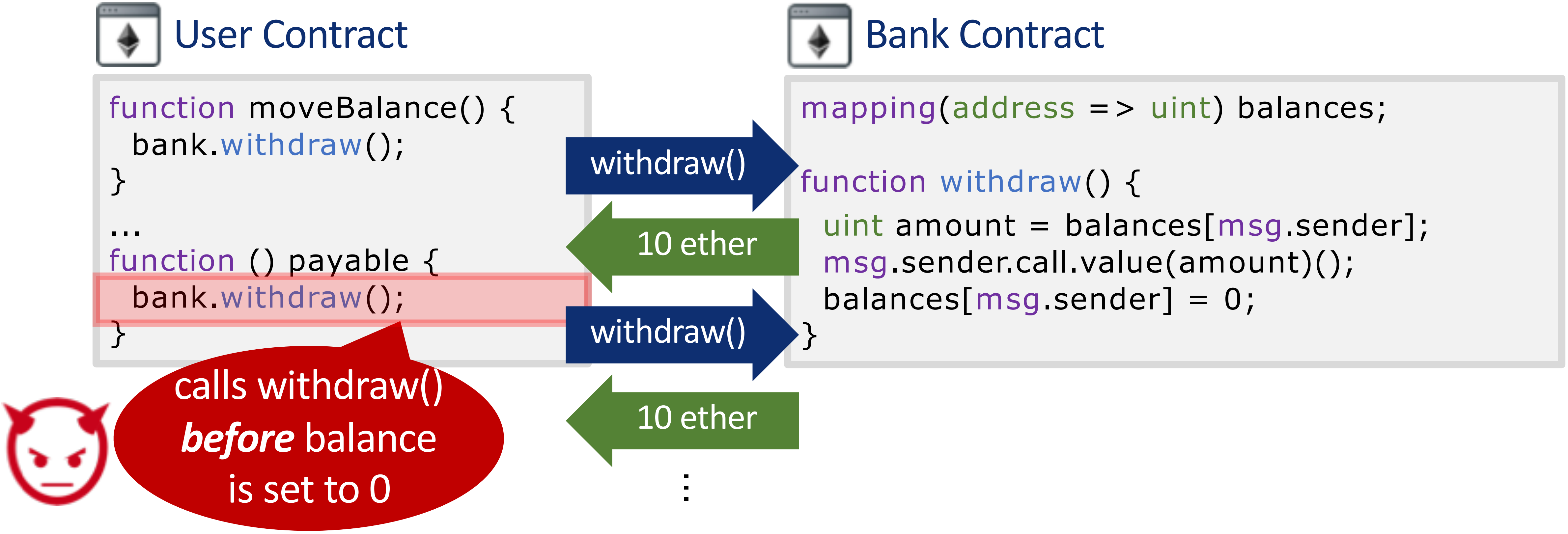
```
ng(address => uint) balances;

) {
  ances[msg.sender];
  lue(amount)();
ces[msg.sender] = 0;
```

calls the default "fallback” function

balance is zeroed *after* ether transfer

Can the user contract withdraw more than its balance?

The DAO hack: Reentrancy



User Contract

```
function moveBalance() {
  bank.withdraw();
}
...
function () payable {
  bank.withdraw();
}
```

Bank Contract

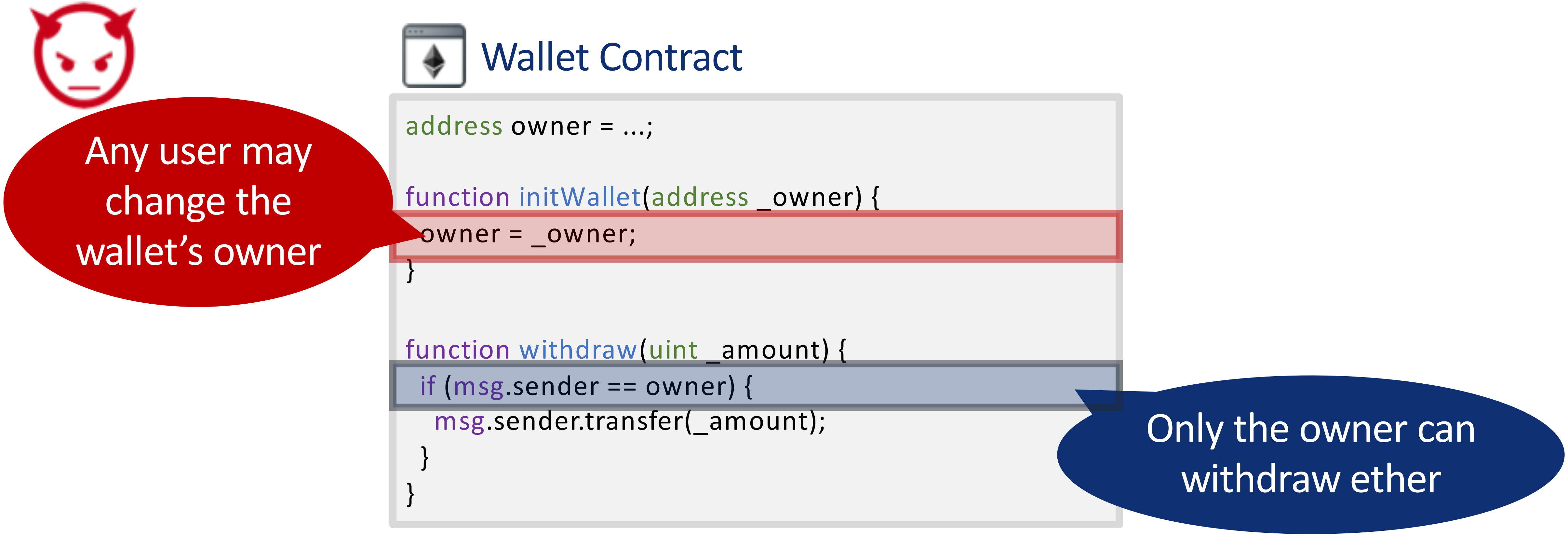
```
mapping(address => uint) balances;

function withdraw() {
  uint amount = balances[msg.sender];
  msg.sender.call.value(amount)();
  balances[msg.sender] = 0;
}
```

calls withdraw() ***before*** balance is set to 0

An attacker used this bug to steal 3.6M ether (> ***1B USD today***)

July 2017: Parity Multisig Bug 1

Parity Multisig **Bug 1**: Unprivileged Write to Storage



Wallet Contract

```
address owner = ...;

function initWallet(address _owner) {
  owner = _owner;
}

function withdraw(uint _amount) {
  if (msg.sender == owner) {
    msg.sender.transfer(_amount);
  }
}
```

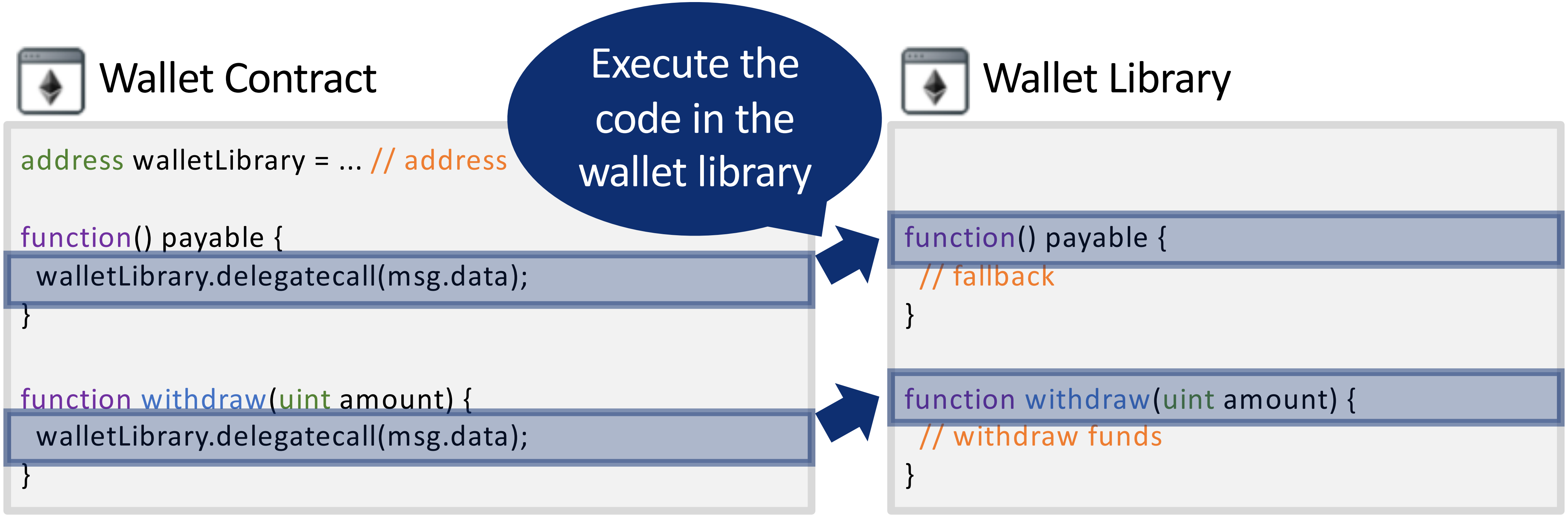
Any user may change the wallet’s owner

Only the owner can withdraw ether

An attacker used a similar bug to **steal $30M** in July 2017

Four months later… Parity Multisig Bug 2

Parity Multisig **Bug 2**: Frozen Wallets



Wallet Contract

```
address walletLibrary = ... // address

function() payable {
  walletLibrary.delegatecall(msg.data);
}

function withdraw(uint amount) {
  walletLibrary.delegatecall(msg.data);
}
```

Execute the code in the wallet library

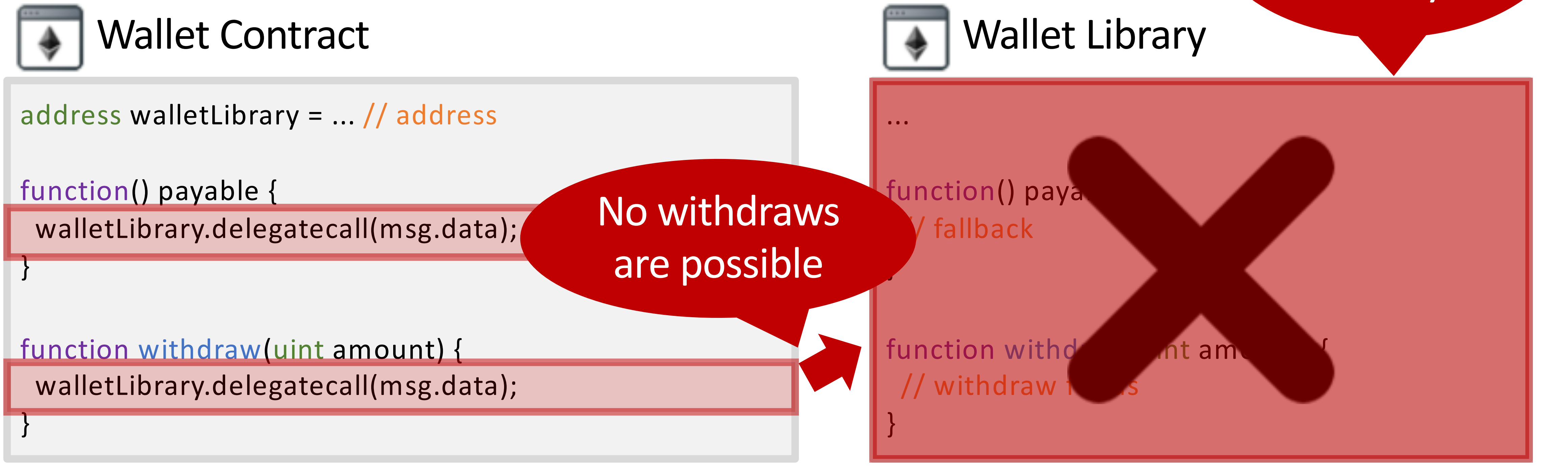
Wallet Library

```
function() payable {
  // fallback
}

function withdraw(uint amount) {
  // withdraw funds
}
```

However, in Ethereum, smart contracts can be killed!

Parity Multisig **Bug 2**: Frozen Wallets

An attacker deleted the library



Wallet Contract

```
address walletLibrary = ... // address

function() payable {
  walletLibrary.delegatecall(msg.data);
}

function withdraw(uint amount) {
  walletLibrary.delegatecall(msg.data);
}
```

No withdraws are possible

Wallet Library

```
...

function() paya
  // fallback
}

function withd     int am     {
  // withdraw f    s
}
```

A user **froze $170M** by deleting the wallet library

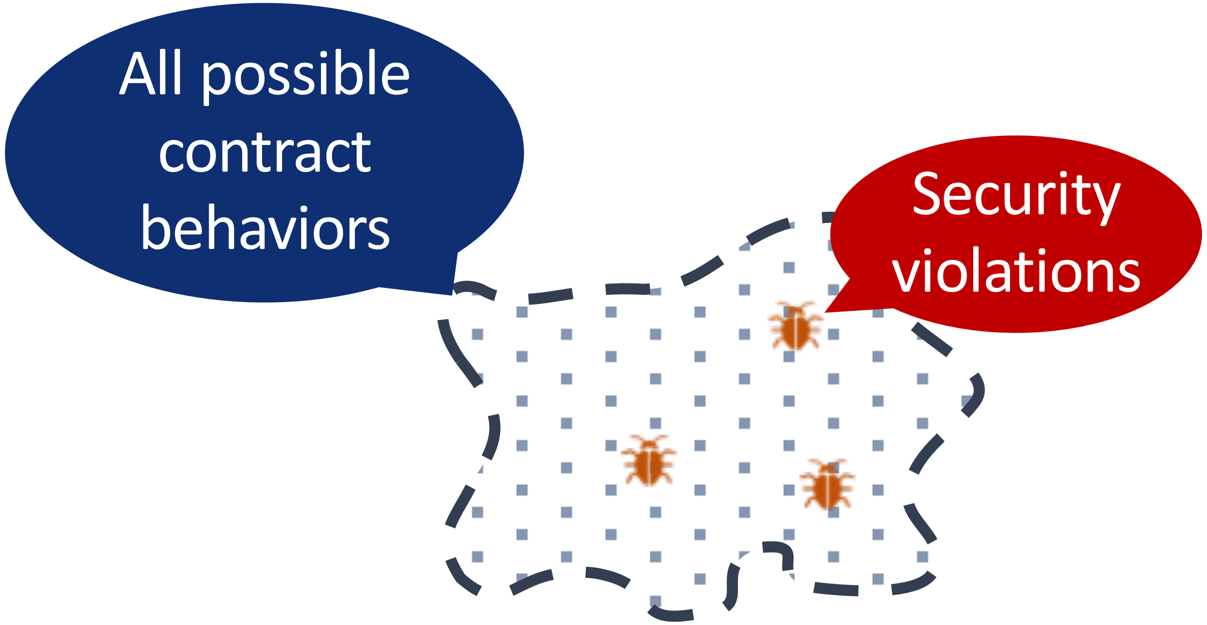
Relevant Security Properties...

- Unexpected ether flows
- Insecure coding, such as unprivileged writes
- Use of unsafe inputs (e.g., reflection, hashing, ...)
- Reentrant method calls (e.g., DAO bug)
- Manipulating ether flows via transaction reordering

Many of these are nontrivial trace-/hyper-properties

Automated Security Analysis of Smart Contracts: Challenges and Gaps

Security Analysis (high-level view)



All possible contract behaviors

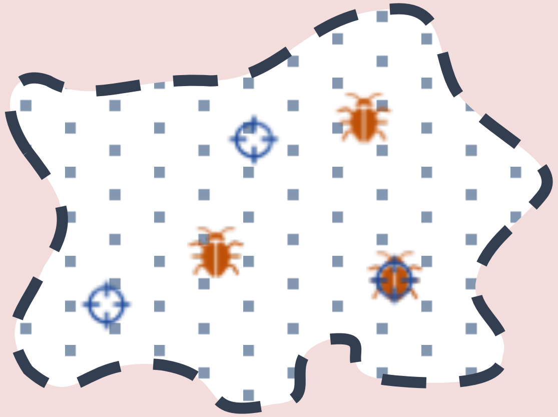
Security violations

Minor issue ☺ : Smart contracts are written in Turing-complete languages

Automated Security Solutions

SECURIFY

Truffle

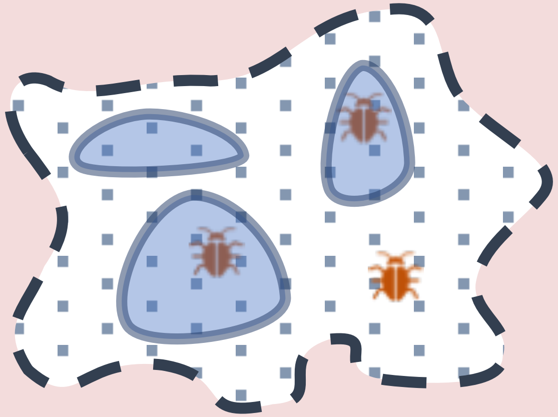


Testing

Report true bugs

Can miss bugs

Oyente, Mythril, MAIAN



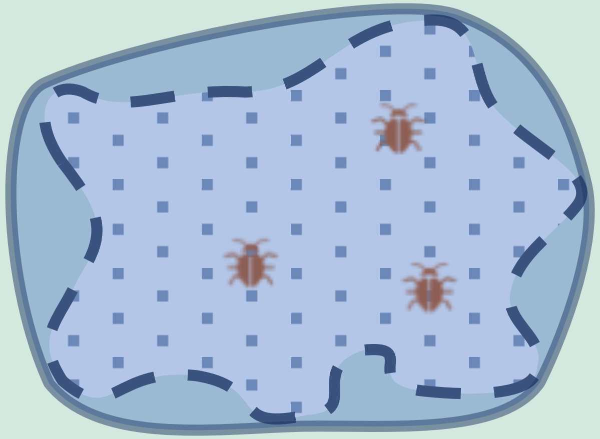
Dynamic (symbolic) analysis

Report true bugs

Can miss bugs

Bug finding

WANTED: Automated Verifier



Can report false alarms

No missed bugs

Verification

Domain-Specific Insight:

When contracts satisfy/violate a property, they often also satisfy/violate a much simpler property

Example: The DAO Hack

Security property

No state changes after call instructions

Hard to verify in general

```
function withdraw() {
  uint amount = balances[msg.sender];
  msg.sender.call.value(amount)();
  balances[msg.sender] = 0;
}
```

Verifies 91% of all deployed contracts

Compliance pattern

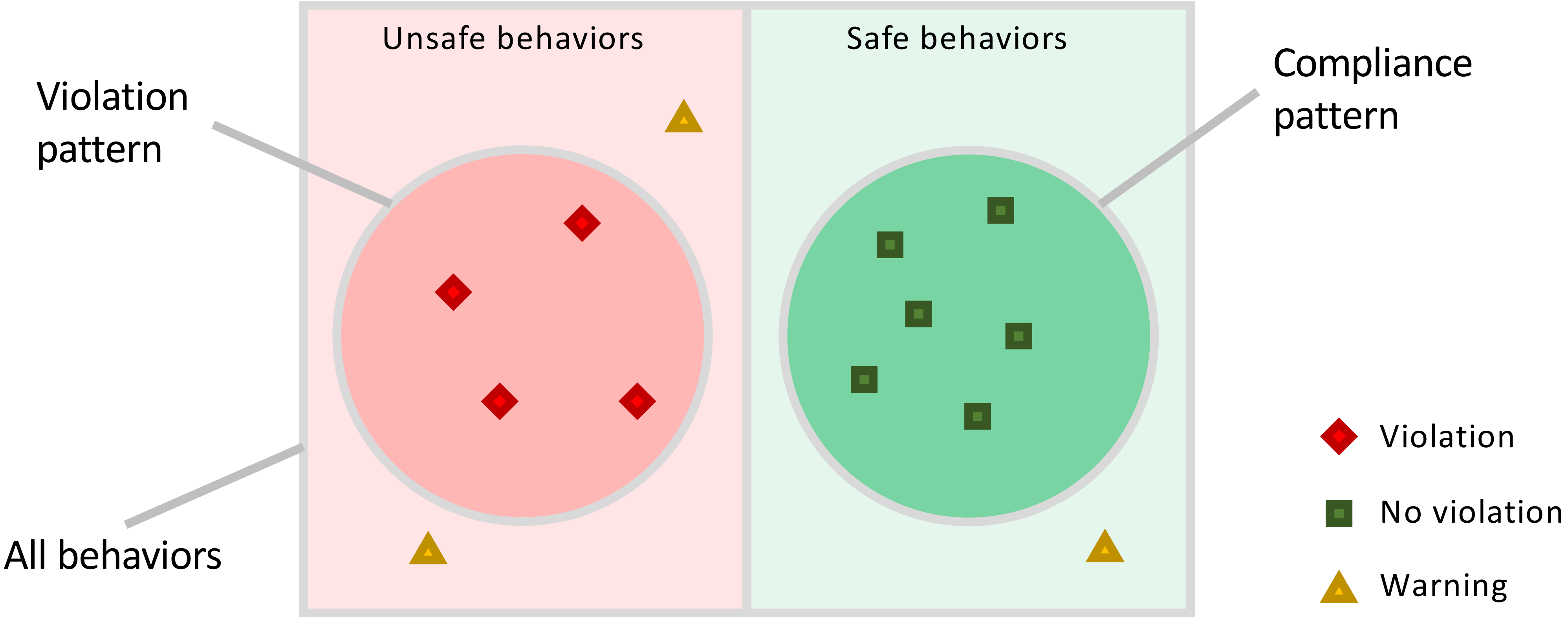
No writes to storage **may follow** call instructions

Violation pattern

A write to storage **must follow** call instructions

Easier to check automatically

Classifying Behaviors using Compliance and Violation Patterns



Unsafe behaviors

Safe behaviors

Violation pattern

Compliance pattern

All behaviors

◆ Violation

■ No violation

▲ Warning

All unsafe behaviors are reported

SECURIFY

www.securify.ch

A practical ***verifier*** for Ethereum smart contracts:

- fully-automated
- extensible
- scalable
- precise
- publicly available

DEMO

Beta version released in Fall 2017

- Regularly used by auditors to perform professional security audits

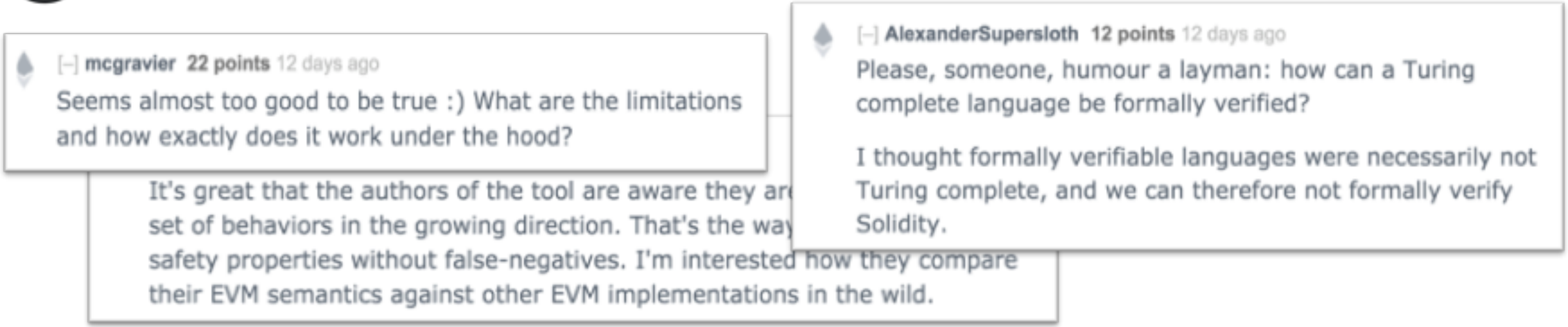
New release coming up very soon

95% positive feedback

> 8K uploaded smart contracts

> 800 users signed up for updates

Interesting discussions on Reddit



[–] mcgravier 22 points 12 days ago

Seems almost too good to be true :) What are the limitations and how exactly does it work under the hood?

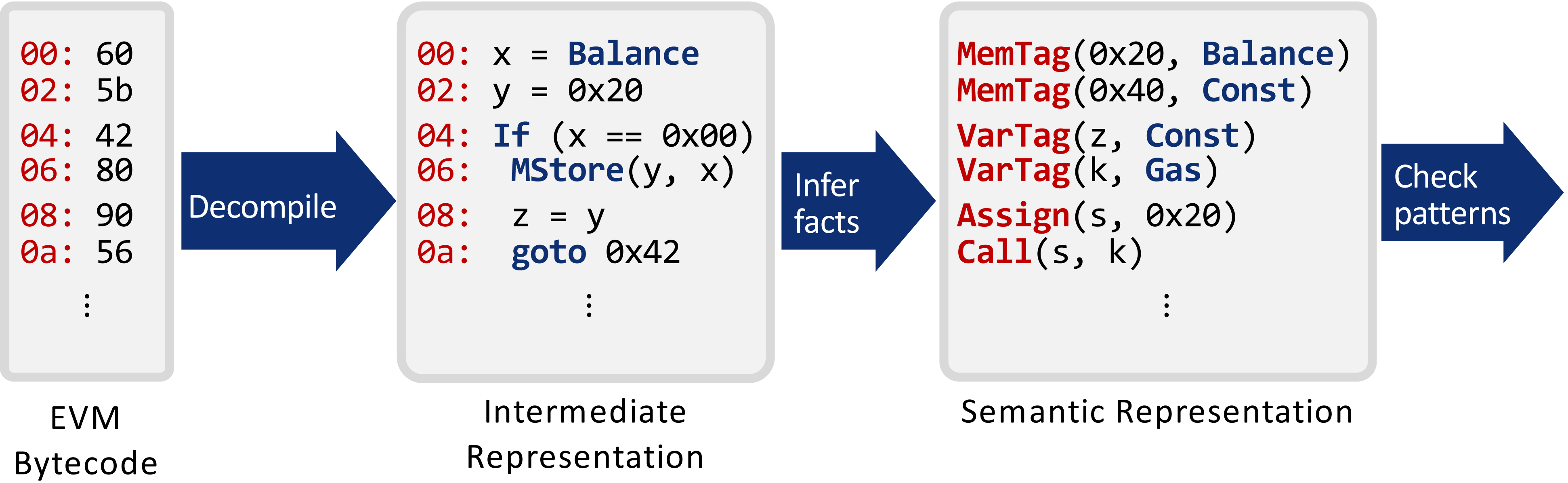
It's great that the authors of the tool are aware they ar
set of behaviors in the growing direction. That's the way
safety properties without false-negatives. I'm interested how they compare
their EVM semantics against other EVM implementations in the wild.

[–] AlexanderSupersloth 12 points 12 days ago

Please, someone, humour a layman: how can a Turing complete language be formally verified?

I thought formally verifiable languages were necessarily not Turing complete, and we can therefore not formally verify Solidity.

Securify: Under the Hood



```
00: 60
02: 5b
04: 42
06: 80
08: 90
0a: 56
 ⋮
```

EVM Bytecode

Decompile

```
00: x = Balance
02: y = 0x20
04: If (x == 0x00)
06:  MStore(y, x)
08:   z = y
0a:  goto 0x42
        ⋮
```

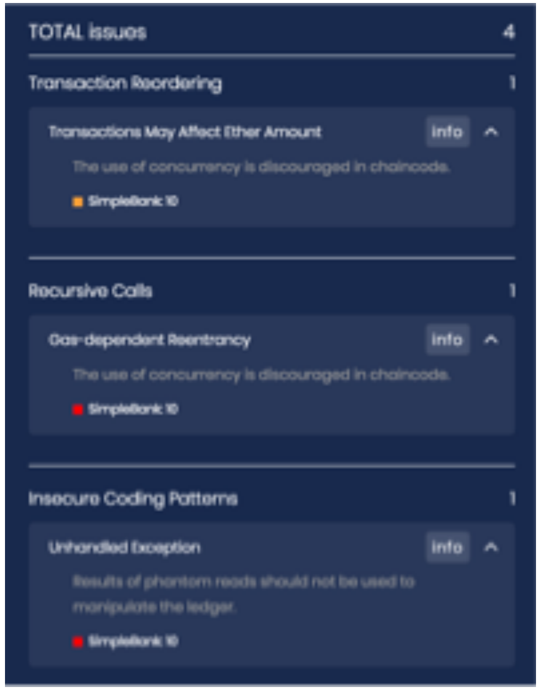
Intermediate Representation

Infer facts

```
MemTag(0x20, Balance)
MemTag(0x40, Const)
VarTag(z, Const)
VarTag(k, Gas)
Assign(s, 0x20)
Call(s, k)
          ⋮
```

Semantic Representation

Check patterns



TOTAL issues 4

Transaction Reordering 1

Transactions May Affect Ether Amount

The use of concurrency is discouraged in chaincode.

🟧 SimpleBank: 10

Recursive Calls 1

Gas-dependent Reentrancy

The use of concurrency is discouraged in chaincode.

🟥 SimpleBank: 10

Insecure Coding Patterns 1

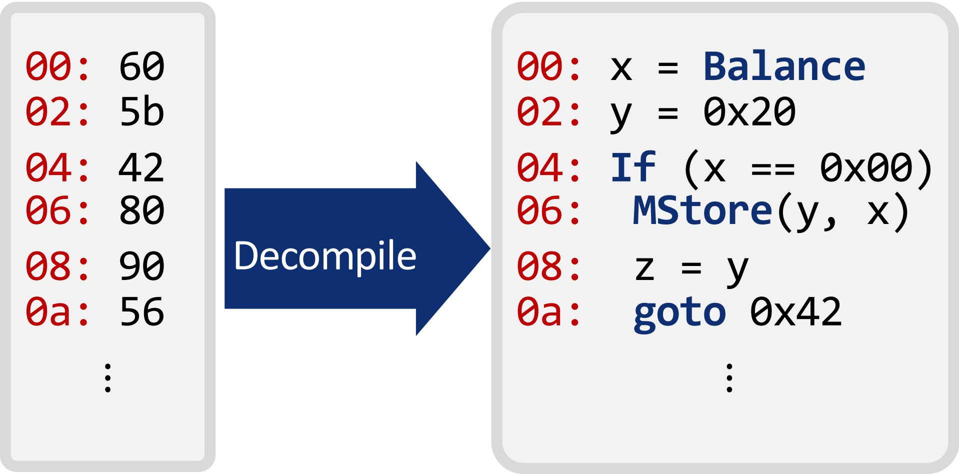
Unhandled Exception

Results of phantom reads should not be used to manipulate the ledger.

🟥 SimpleBank: 10

Fully automated, sound, scalable, extensible

Securify: Under the Hood



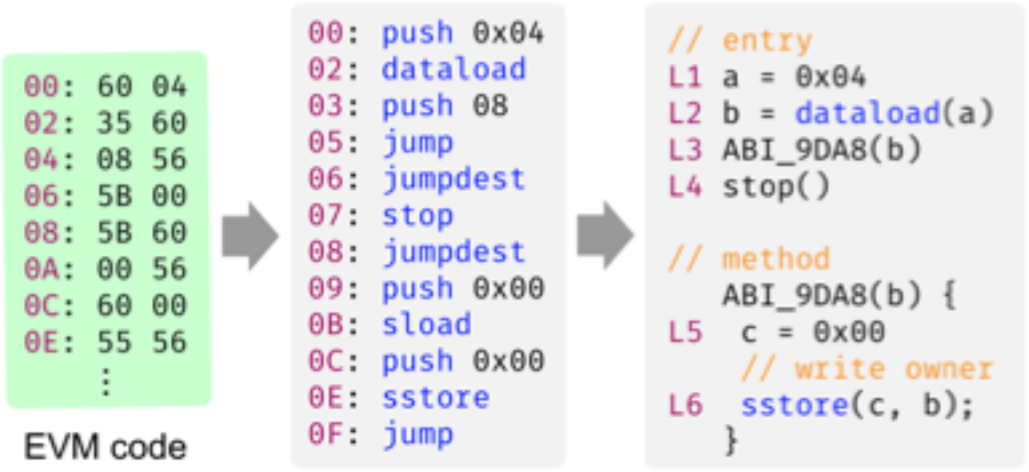
```
00: 60
02: 5b
04: 42
06: 80
08: 90
0a: 56
     ⋮
```

```
00: x = Balance
02: y = 0x20
04: If (x == 0x00)
06:  MStore(y, x)
08:  z = y
0a:  goto 0x42
         ⋮
```

EVM Bytecode

Intermediate Representation

From EVM to CFG over SSA



```
00: 60 04
02: 35 60
04: 08 56
06: 5B 00
08: 5B 60
0A: 00 56
0C: 60 00
0E: 55 56
  ⋮
```

EVM code

```
00: push 0x04
02: dataload
03: push 08
05: jump
06: jumpdest
07: stop
08: jumpdest
09: push 0x00
0B: sload
0C: push 0x00
0E: sstore
0F: jump
```

Parsed code

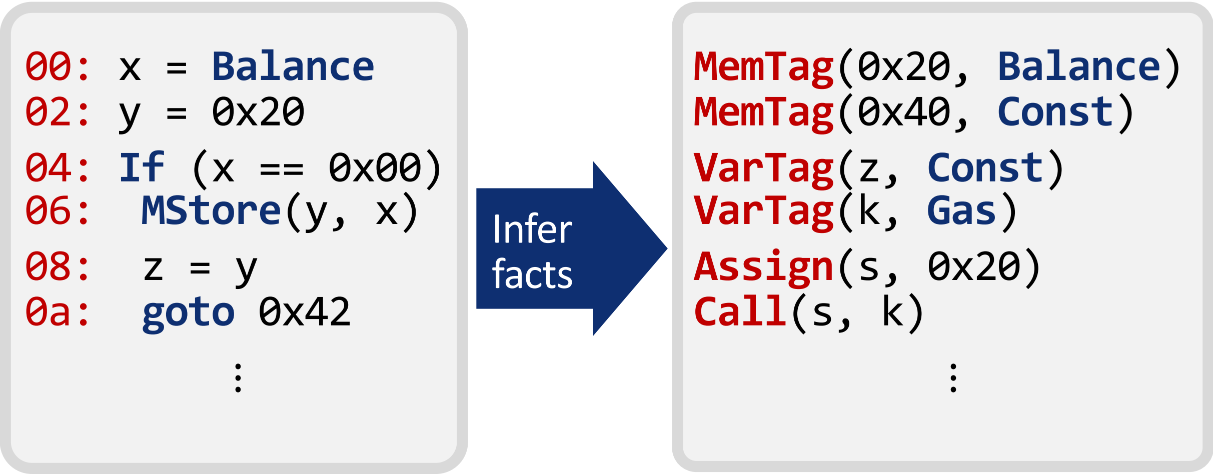
```
// entry
a = 0x04
b = dataload(a)
ABI_9DA8(b)
stop()

// method
ABI_9DA8(b) {
 c = 0x00
 // write owner
 sstore(c, b);
}
```

Decompiled code

Decompiling EVM bytecode:

- Convert into **static single assignment form** (each variable is assigned once)
- Perform **partial evaluation** (to resolve jump destination, memory/storage offsets)
- Identify and inline methods (to enable context-sensitive analysis)
- Construct **control-flow graph**

Securify: Under the Hood



```
00: x = Balance
02: y = 0x20
04: If (x == 0x00)
06:  MStore(y, x)
08:  z = y
0a:  goto 0x42
        ⋮
```

Intermediate Representation

```
MemTag(0x20, Balance)
MemTag(0x40, Const)
VarTag(z, Const)
VarTag(k, Gas)
Assign(s, 0x20)
Call(s, k)
        ⋮
```

Semantic Representation

Which facts are relevant for verifying smart contracts?

Semantic Facts

Many properties can be checked on the contract’s dependency graph

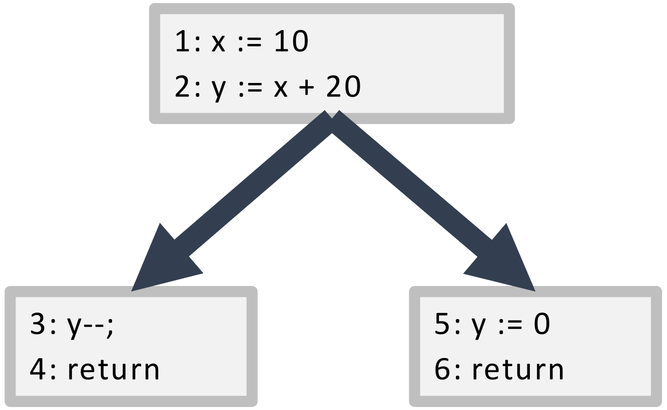
Flow dependencies	
$MayFollow(l, l')$	The instruction at label l may follow that at label l'
$MustFollow(l, l')$	The instruction at label l must follow that at label l'
Data dependencies	
$MayDepOn(x, t)$	The value of x may depend on tag t
$DetBy(x, t)$	For different values of t the value of x is different.

A tag can be an instruction (e.g. Caller) or a variable

The inference of all semantic facts is declaratively specified in Datalog

Example: $MayFollow$

$$MayFollow(i,j) \leftarrow Follow(i,j)$$
$$MayFollow(i,j) \leftarrow Follow(i,k), MayFollow(k,j)$$



```
1: x := 10
2: y := x + 20
```

```
3: y--;
4: return
```

```
5: y := 0
6: return
```

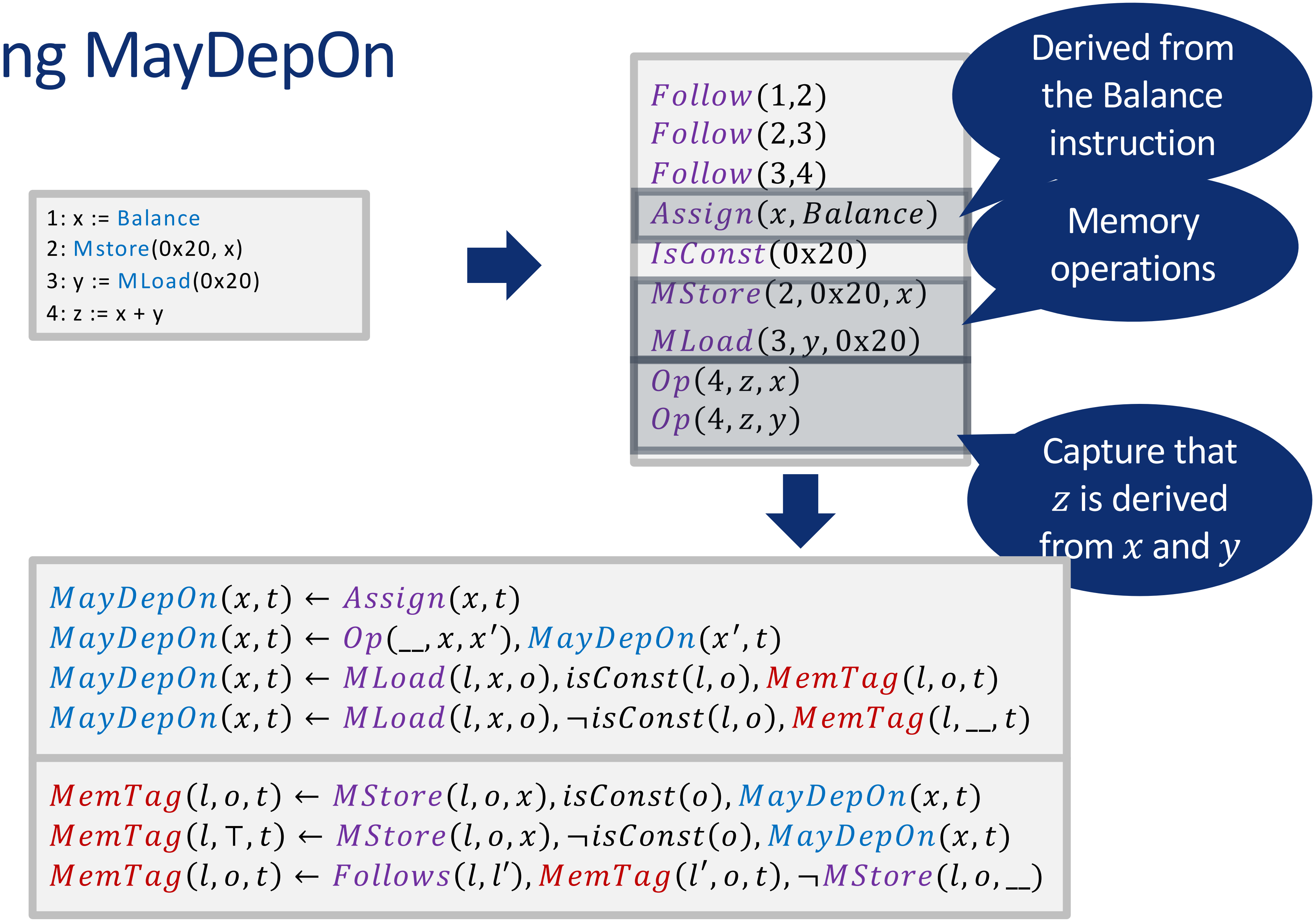
$Follow(1,2)$
$Follow(2,3)$
$Follow(3,4)$
$Follow(2,5)$
$Follow(5,6)$

Datalog input

$MayFollow(1,2)$
$MayFollow(1,3)$
$MayFollow(1,4)$
$MayFollow(1,5)$
$MayFollow(1,6)$
$MayFollow(2,3)$
$MayFollow(2,4)$
$MayFollow(2,5)$
$MayFollow(2,6)$
$MayFollow(3,4)$
$MayFollow(5,6)$

Datalog fixpoint

Deriving MayDepOn



```
1: x := Balance
2: Mstore(0x20, x)
3: y := MLoad(0x20)
4: z := x + y
```

$$\begin{array}{l}
\mathit{Follow}(1{,}2)\\
\mathit{Follow}(2{,}3)\\
\mathit{Follow}(3{,}4)\\
\mathit{Assign}(x, \mathit{Balance})\\
\mathit{IsConst}(0\mathrm{x}20)\\
\mathit{MStore}(2, 0\mathrm{x}20, x)\\
\mathit{MLoad}(3, y, 0\mathrm{x}20)\\
\mathit{Op}(4, z, x)\\
\mathit{Op}(4, z, y)
\end{array}$$

Derived from the Balance instruction

Memory operations

Capture that z is derived from x and y

$$\begin{array}{l}
\mathit{MayDepOn}(x, t) \leftarrow \mathit{Assign}(x, t)\\
\mathit{MayDepOn}(x, t) \leftarrow \mathit{Op}(\_\_, x, x'), \mathit{MayDepOn}(x', t)\\
\mathit{MayDepOn}(x, t) \leftarrow \mathit{MLoad}(l, x, o), \mathit{isConst}(l, o), \mathit{MemTag}(l, o, t)\\
\mathit{MayDepOn}(x, t) \leftarrow \mathit{MLoad}(l, x, o), \neg \mathit{isConst}(l, o), \mathit{MemTag}(l, \_\_, t)
\end{array}$$

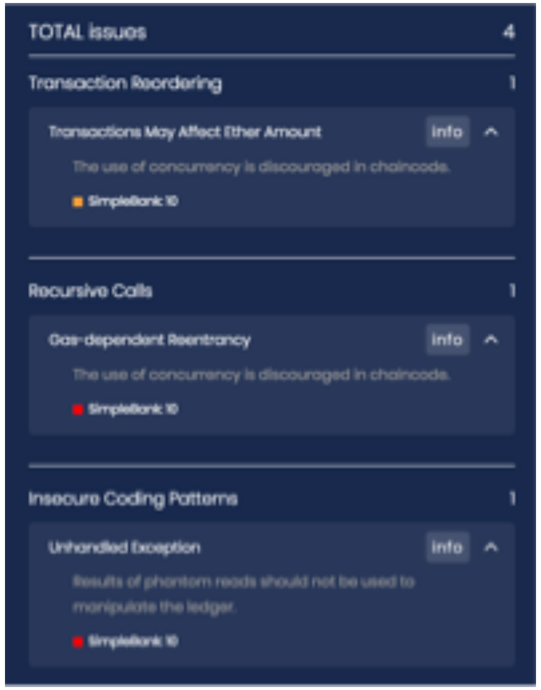
$$\begin{array}{l}
\mathit{MemTag}(l, o, t) \leftarrow \mathit{MStore}(l, o, x), \mathit{isConst}(o), \mathit{MayDepOn}(x, t)\\
\mathit{MemTag}(l, \top, t) \leftarrow \mathit{MStore}(l, o, x), \neg \mathit{isConst}(o), \mathit{MayDepOn}(x, t)\\
\mathit{MemTag}(l, o, t) \leftarrow \mathit{Follows}(l, l'), \mathit{MemTag}(l', o, t), \neg \mathit{MStore}(l, o, \_\_)
\end{array}$$

Securify: Under the Hood

```
MemTag(0x20, Balance)
MemTag(0x40, Const)
VarTag(z, Const)
VarTag(k, Gas)
Assign(s, 0x20)
Call(s, k)
                ⋮
```

Semantic Representation

Check patterns



TOTAL issues 4

Transaction Reordering 1

Transactions May Affect Ether Amount

The use of concurrency is discouraged in chaincode.

🟧 SimpleBank:10

Recursive Calls 1

Gas-dependent Reentrancy

The use of concurrency is discouraged in chaincode.

🟥 SimpleBank:10

Insecure Coding Patterns 1

Unhandled Exception

Results of phantom reads should not be used to manipulate the ledger.

🟥 SimpleBank:10

Patterns DSL

(*Labels*)	l ::= (labels)
(*Vars*)	x ::= (variables)
(*Tags*)	$t ::= l \mid x$
(*Instr*)	$n ::= Instr(l, x, \dots, x)$
(*Facts*)	$f ::= MayFollow(l, l) \mid MustFollow(l, l)$
	$\mid MayDepOn(x, t) \mid DetBy(x, t)$
(*Patterns*)	$p ::= f \mid \forall\, n{:}\, p \mid \exists n{:}\, p \mid p \wedge p \mid \neg p$

Detecting the DAO Hack

```
function withdraw() {
  uint amount = balances[msg.sender];
  msg.sender.call.value(amount)();
  balances[msg.sender] = 0;
}
```

Call instruction followed by a write to storage

Formalized as a trace property

Security property: No state changes after call instructions

Compliance pattern $Call(l,\_,\_,\_): \neg\exists SStore(l',\_,\_).\,MayFollow(l,l')$

Violation pattern $Call(l,\_,\_,\_): \exists SStore(l',\_,\_).\,MustFollow(l,l')$

Proofs establish a formal logical relation between the property and its patterns

Detecting Unrestricted Writes

```
address owner = …;

function initWallet(address _owner) {
  owner = _owner;
}
```

Unrestricted write

Formalized as a hyperproperty

Security property: No storage offset is writable by all users

Compliance pattern $SStore(\_, x, \_): DetBy(x, Caller)$

Violation pattern $SStore(l, x, \_\_): \neg MayDepOn(x, Caller) \wedge \neg MayDepOn(l, Caller)$

How well does this approach work in practice?

Securify vs. Existing Solutions

State-of-the-art security checkers for Ethereum smart contracts

- Oyente
- Mythril

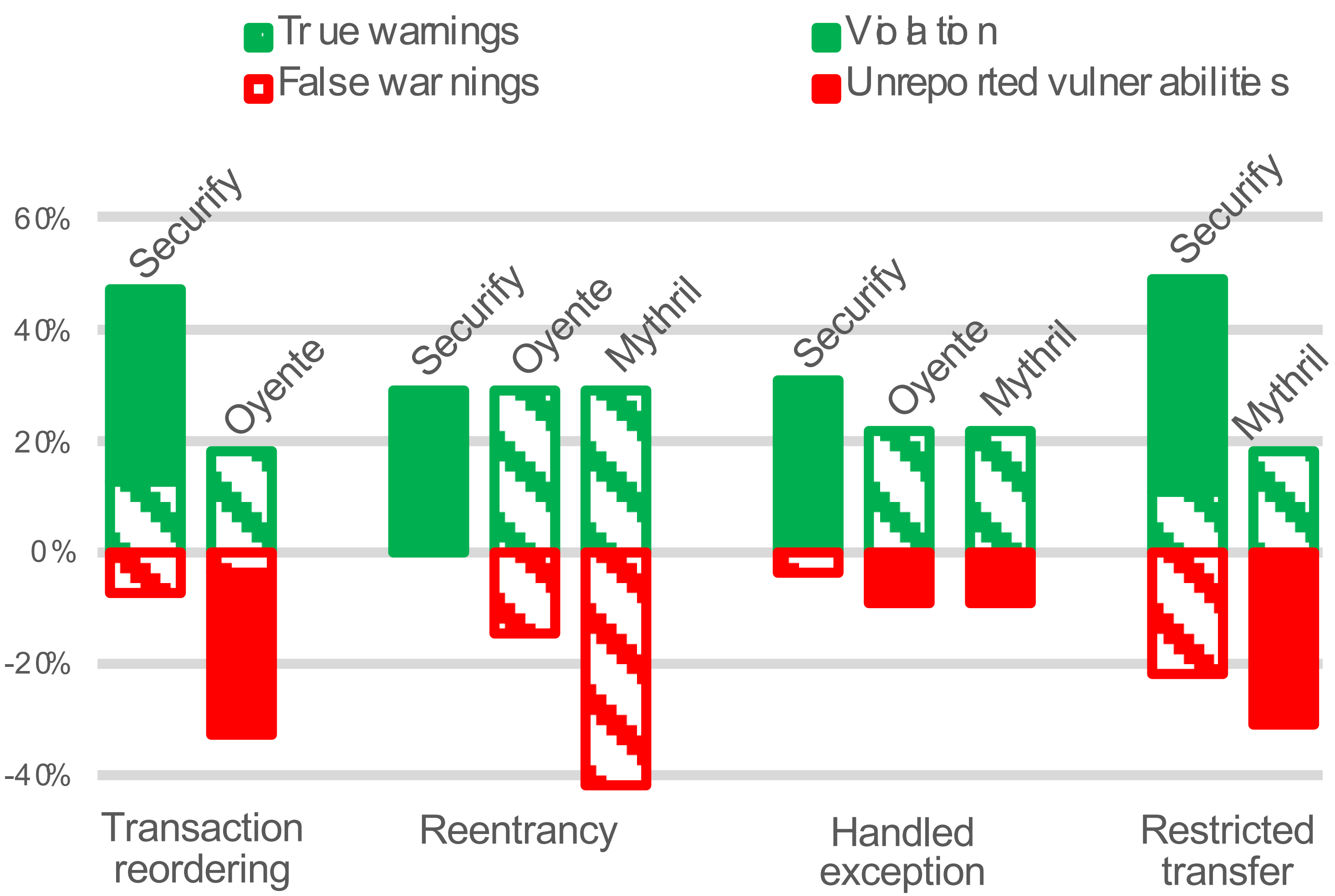
Dataset

- 80 open-source smart contracts

Experiment

- Run contracts using Securify, Oyente, and Mythril
- Manually inspect each reported vulnerability

Securify vs. Oyente vs. Mythril



Research

SRL
SECURE, RELIABLE, INTELLIGENT SYSTEMS LAB

ICE center@ ETH

AI² http://ai2.ethz.ch

SECURIFY http://securify.ch

DEGUARD http://apk-deguard.com

JS NICE http://jsnice.org

PSI SOLVER http://psisolver.org

EVENT RACER http://eventracer.org

Start-ups

CHAINSECURITY

Securing the blockchain

https://chainsecurity.com

WE'RE HIRING **jobs@chainsecurity.com**

✉ contact@chainsecurity.com

@chain_security