Securify: Practical Security Analysis of Smart Contracts

https://securify.ch

Petar Tsankov

Andrei Dan

Dana Drachsler-Cohen

Arthur Gervais

Florian Bünzli

Martin Vechev

Grant: ethereum foundation

Startup: CHAINSECURITY

What is a smart contract?

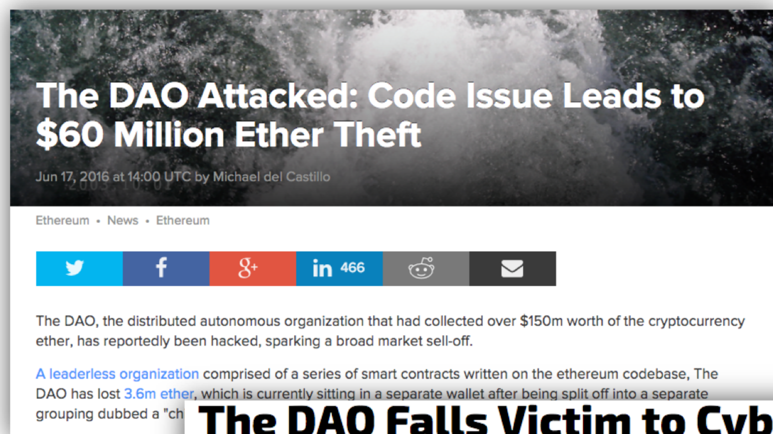
```
mapping(address => uint) balances;

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

Transfer funds to the caller

- Small programs that handle cryptocurrencies
- Written in high-level languages (e.g., Solidity)
- Usually no patching after release

What can go wrong when programs handle billions worth of USD?

Smart contract *bugs* in the news



The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

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

Ethereum • News • Ethereum

in 466

The DAO, the distributed autonomous organization that had collected over $150m worth of the cryptocurrency ether, has reportedly been hacked, sparking a broad market sell-off.

A leaderless organization comprised of a series of smart contracts written on the ethereum codebase, The DAO has lost 3.6m ether, which is currently sitting in a separate wallet after being split off into a separate grouping dubbed a "ch

The DAO Falls Victim to Cyber Attack Leading Ethereum to Crash Over 20%

The event is still ongoing as hackers have already stolen over 3.5 million ETH from the DAO's coffers.

Avi Mizrahi | Trading (CryptoCurrency) | Friday, 17/06/2016|12:45 GMT

Photo: Finance Magnates

Share this article

Smart contract *bugs* in the news

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

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

Ethereum • News • Ethereum

in 466

The DAO, the distributed autonomous organization that had collected over $150m worth of the cryptocurrency ether, has reportedly been hacked, sparking a broad market sell-off.

A leaderless organization comprised of a series of smart contracts written on the ethereum codebase, The DAO has lost 3.6m ether, which is currently sitting in a separate wallet after being split off into a separate grouping dubbed a "ch

The DAO Falls Victim to Cyber Attack Leading Ethereum to Crash Over 20%

The event is still ongoing as hackers have already stolen over 3.5 million ETH from the DAO's coffers.

Avi Mizrahi | Trading (CryptoCurrency) | Friday, 17/06/2016|12:45 GMT

Photo: Finance Magnates

Share this article

CNBC

Search Quotes, News & Video

CYBERSECURITY

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

$32 million worth of digital currency ether stolen by hackers

- Around 153,000 ether tokens worth $32.6 million were taken by hackers on Wednesday.
- A vulnerability in Parity's multisignature wallet was exploited by hackers.
- This latest theft follows an incident on Monday where $7 million worth of ether tokens were stolen.

Luke Graham | @LukeWGraham

Published 7:41 AM ET Thu, 20 July 2017 | Updated 10:51 AM ET Thu, 20 July 2017

CNBC

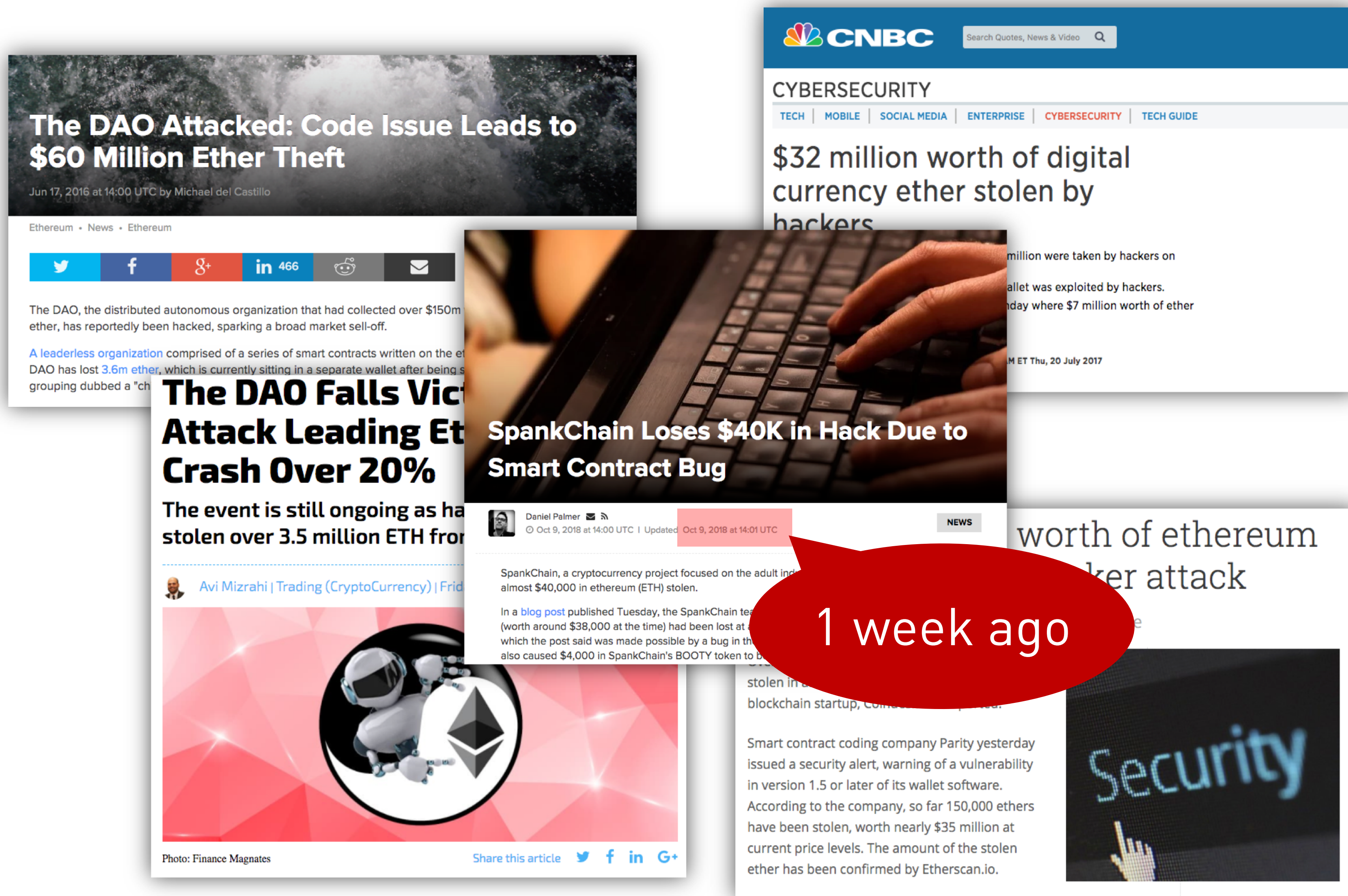
Over $30 million worth of ethereum stolen in another hacker attack

Theft due to security issue with Parity's wallet software

Over $30 million worth of ethereum have been stolen in another hacking attack targeting a blockchain startup, Coindesk has reported.

Smart contract coding company Parity yesterday issued a security alert, warning of a vulnerability in version 1.5 or later of its wallet software. According to the company, so far 150,000 ethers have been stolen, worth nearly $35 million at current price levels. The amount of the stolen ether has been confirmed by Etherscan.io.

Smart contract *bugs* in the news



June 2016: The DAO hack

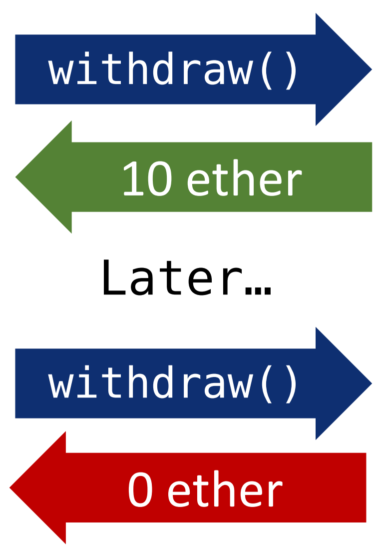
The DAO hack

User Contract

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

DAO Contract

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

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



The DAO hack

User Contract

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

DAO Contract

```
mapping(address =>

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

calls the default "fallback" function

balance is zeroed ***after*** transfer

withdraw()

10 ether

Later…

withdraw()

0 ether

The DAO hack

User Contract

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

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

DAO Contract

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

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

withdraw() →

← 10 ether

withdraw() →

← 10 ether

⋮

Many critical vulnerabilities

In 2017, more than

$300M

have been lost due to these issues

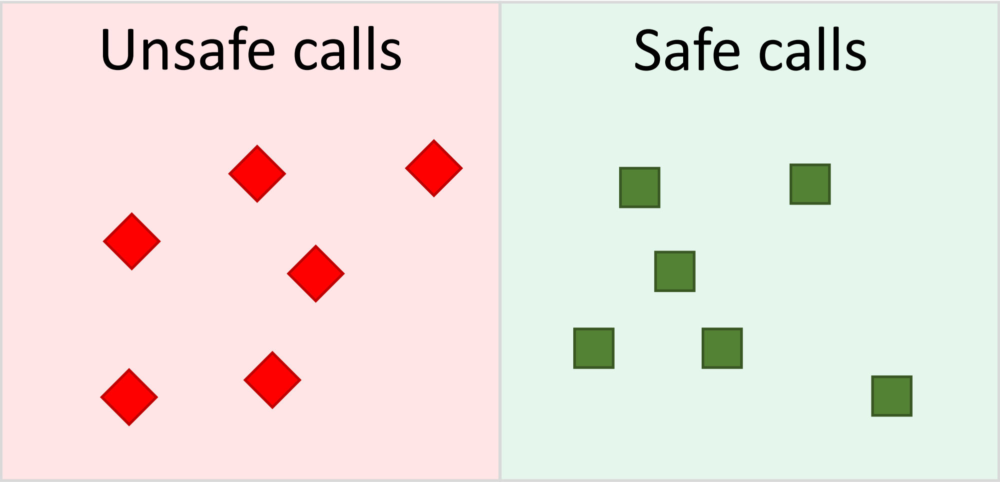
- Unexpected ether flows
- Unprivileged writes
- Use of unsafe inputs
- Reentrant method calls
- Transaction reordering

Wanted: Automated security analysis

The DAO hack

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

Security property: No state changes after call instructions

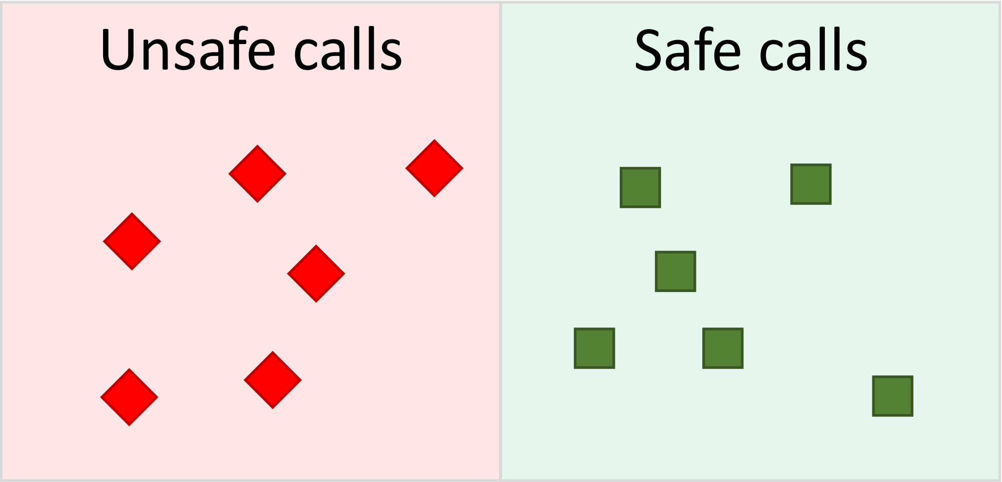


Can we automatically find all unsafe calls?

The DAO hack

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

Security property: No state changes after call instructions



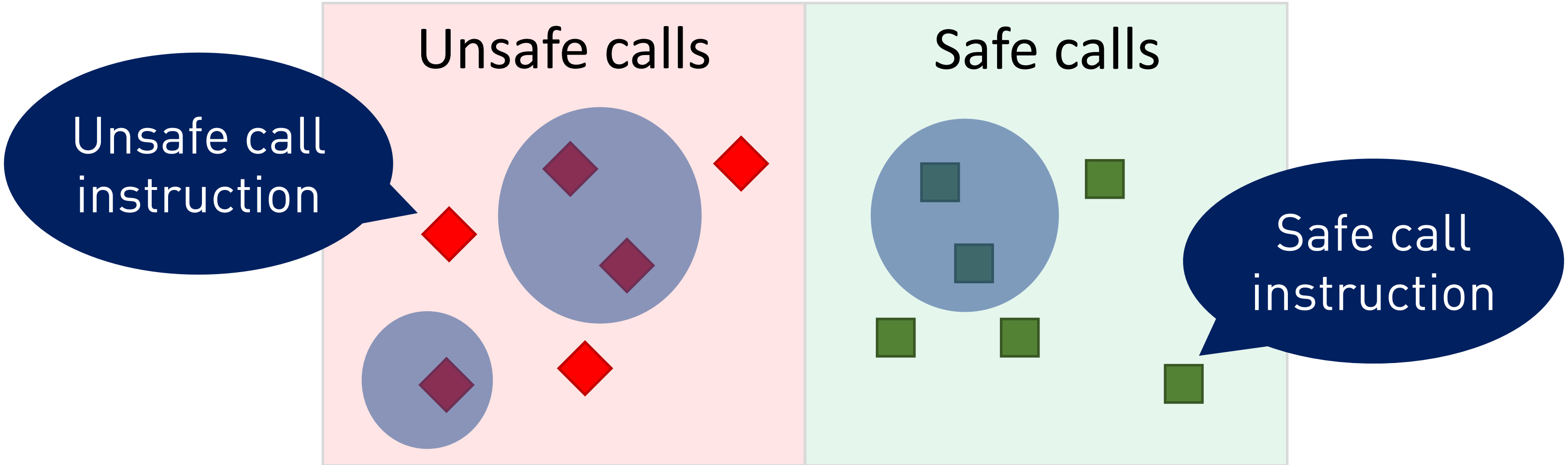
Can we automatically find all unsafe calls?

No, smart contracts are Turing-complete

The DAO hack

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

Security property: No state changes after call instructions



Can we automatically find all unsafe calls?

No, smart contracts are Turing-complete

Existing solutions focus on bug finding and can **miss issues**

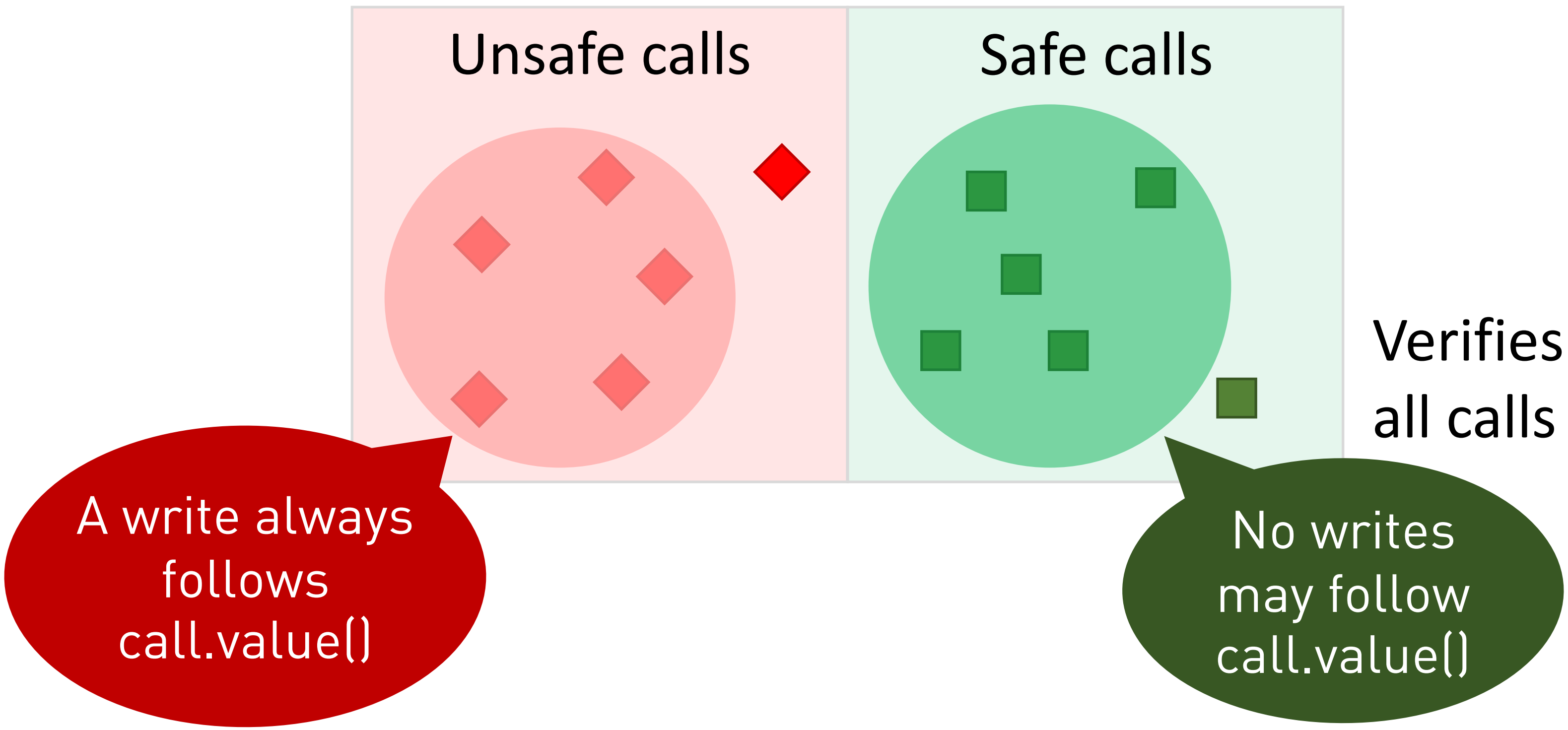
Insight

When contracts satisfy/violate a security property, they often satisfy/violate a simpler property

The DAO hack

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

Security property: No state changes after call instructions



Verifies **91%** of all calls

Violation pattern

Compliance pattern

SECURIFY

www.securify.ch

Scalable and fully ***automated verifier*** for Ethereum smart contracts

Impact

SECURIFY

Used daily by security auditors
(29K+ contracts scanned so far)

1K+ subscribers

Grants: ethereum foundation

New startup: CHAINSECURITY

Securify: System overview

```
push 0x04
dataload
push 0x08
jump
jumpdest
stop
jumpdest
  ⋮
```

EVM bytecode

1. decompile

```
1: a = 0x04
2: b = load(a)
3: abi_00(b)
4: stop
   abi_00(b)
5:    c = 0x00
6:    sstore(c,b)
         ⋮
```

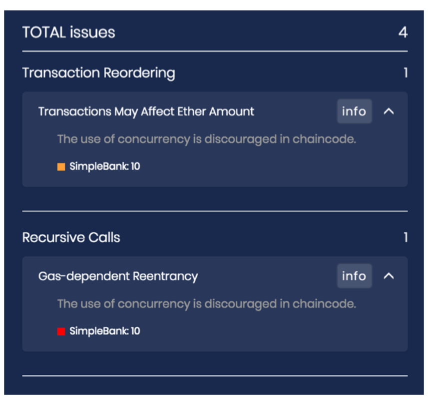
Intermediate representation

2. infer facts

```
assign(1, a, 0x04)
follow(2, 1)
mayDepOn(b, a)
load(2, b, a)
follow(3,2)
follow(5,3)
         ⋮
```

Semantic representation

3. check patterns



Security report

Step 1: Decompilation

```
push 0x04
dataload
push 0x08
jump
jumpdest
stop
jumpdest
    ⋮
```

EVM bytecode

1. decompile

```
1: a = 0x04
2: b = load(a)
3: abi_00(b)
4: stop
   abi_00(b)
5:     c = 0x00
6:     sstore(c,b)
          ⋮
```

Intermediate representation

- Static single assignment form
- Control-flow graph recovery

Step 2: Inferring semantic facts

```
1: a = 0x04
2: b = load(a)
3: abi_00(b)
4: stop
   abi_00(b)
5:    c = 0x00
6:    sstore(c,b)
          ⋮
```

Intermediate representation

2. infer facts

```
assign(1, a, 0x04)
follow(2, 1)
mayDepOn(b, a)
load(2, b, a)
follow(3,2)
follow(5,3)
          ⋮
```

Semantic representation

Step 2: Inferring semantic facts

Scalable inference of semantic facts using Datalog solvers

Datalog program

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

```
1: a = 0x04
2: b = load(a)
3: abi_00(b)
4: stop
   abi_00(b)
5:     c = 0x00
6:     sstore(c,b)
          ⋮
```

IR

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

Datalog input

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

Datalog fixpoint

Step 2: Inferring semantic facts

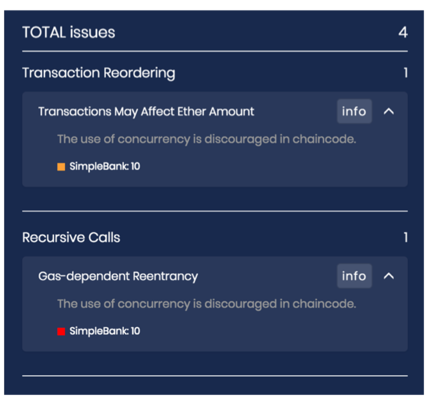
Relevant semantic facts

Control-flow analysis	
$mayFollow(L_1, L_2)$	Instruction at label L_1 may follow that at label L_2
$mustFollow(L_1, L_2)$	Instruction at label L_1 must follow that at label L_2
Data-flow analysis	
$mayDepOn(X, T)$	The value of X may depend on tag T
$eq(X, T)$	The values of X and T are equal
$detBy(X, T)$	For different values of T the value of X is different

For real-world contracts, Securify infers 1 - 10M such facts

IR Datalog input Datalog fixpoint

Step 3: Check patterns



Security report

3. check patterns

```
assign(1, a, 0x04)
follow(2, 1)
mayDepOn(b, a)
load(2, b, a)
follow(3,2)
follow(5,3)
⋮
```

Semantic representation

Security patterns language

A *pattern* is a logical formula over semantic predicates:

$$\varphi ::= instr(L, Y, X, \dots, X)$$
$$\mid\ eq(\mathrm{X}, \mathrm{T}) \mid detBy(\mathrm{X}, \mathrm{Y}) \mid mayDepOn(\mathrm{X}, \mathrm{Y})$$
$$\mid follow(L, L) \mid\ mayFollow(L, L) \mid mustFollow(L, L)$$
$$\mid \exists X.\varphi \mid \exists L.\varphi \mid \exists T.\varphi \mid \neg\varphi \mid \varphi \wedge \varphi$$

see paper for details

Example: No writes after calls

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

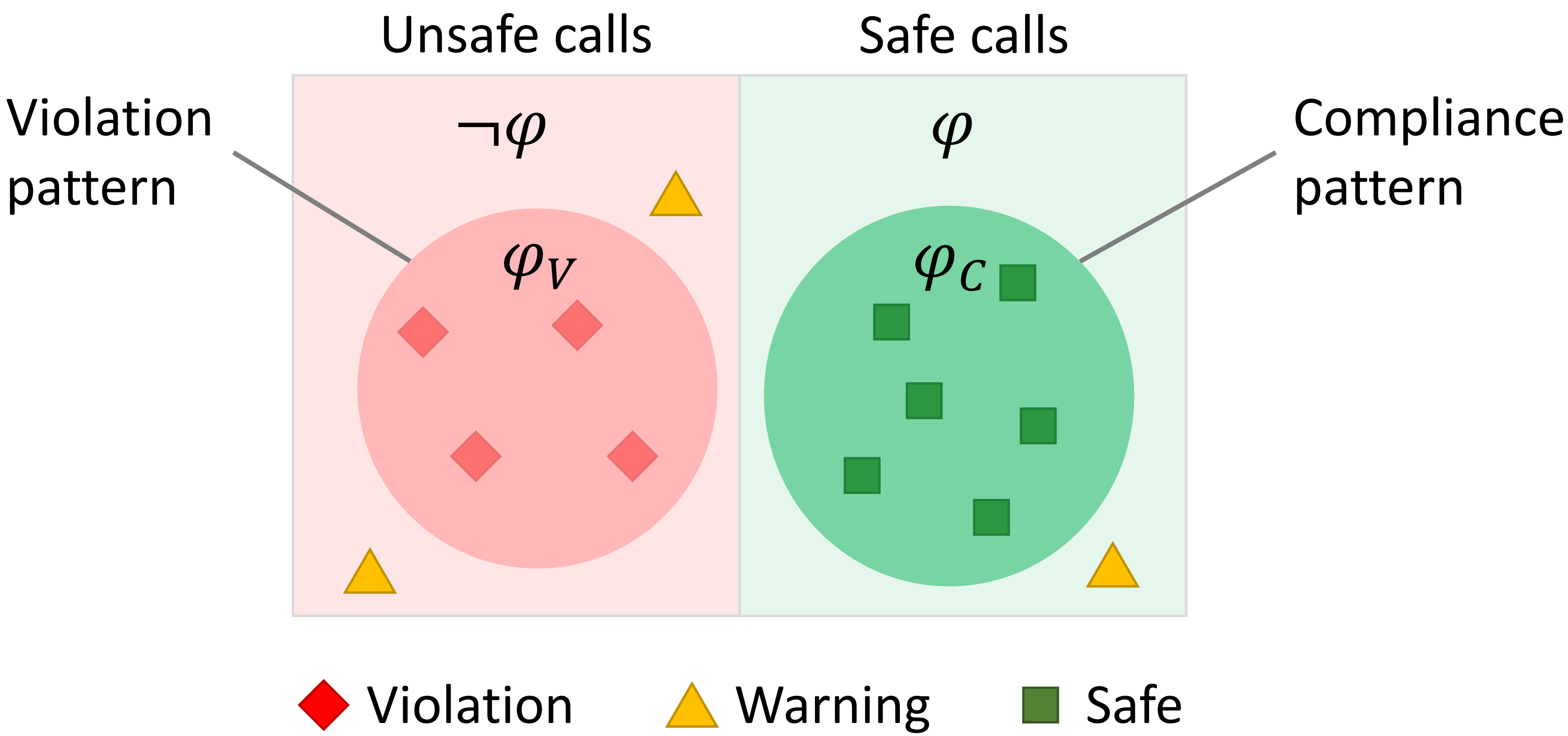
Security property: $\varphi \equiv$ "No state changes after call instructions"

Compliance pattern $\varphi_C \equiv \forall\, call(L_1, \_, \_).\, \neg\exists\, sstore(L_2, \_, \_).\, mayFollow(L_2, L_1)$

Violation pattern $\varphi_C \equiv \exists\, call(L_1, \_, \_).\, \exists\, sstore(L_2, \_, \_).\, mustFollow(L_2, L_1)$

We can (manually) prove that: $\varphi_C \Rightarrow \varphi$ and $\varphi_V \Rightarrow \neg\varphi$

Security report



All unsafe calls are reported as either violations or warnings

Patterns for relevant security properties

Property	Type	Security Pattern
LQ: Ether liquidity	*compliance*	$\mathit{all}\ \mathsf{stop}(L_1).\ \mathit{some}\ \mathsf{goto}(L_2, X, L_3).\ X = \mathsf{callvalue} \land \mathit{Follow}(L_2, L_4) \land L_3 \neq L_4 \land \mathit{MustFollow}(L_4, L_1)$
	compliance	$\mathit{some}\ \mathsf{call}(L_1, \_, \_, \mathit{Amount}).\mathit{Amount} \neq 0 \lor \mathit{DetBy}(\mathit{Amount}, \mathsf{data})$
	violation	$(\mathit{some}\ \mathsf{stop}(L).\ \neg \mathit{MayDepOn}(L, \mathsf{callvalue})) \ \land\ (\mathit{all}\ \mathsf{call}(\_, \_, \_, \mathit{Amount}).\ \mathit{Amount} = 0)$
NW: No writes after call	*compliance*	$\mathit{all}\ \mathsf{call}(L_1, \_, \_, \_).\ \mathit{all}\ \mathsf{sstore}(L_2, \_, \_).\ \neg \mathit{MayFollow}(L_1, L_2)$
	violation	$\mathit{some}\ \mathsf{call}(L_1, \_, \_, \_).\ \mathit{some}\ \mathsf{sstore}(L_2, \_, \_).\ \mathit{MustFollow}(L_1, L_2)$
RW: Restricted write	*compliance*	$\mathit{all}\ \mathsf{sstore}(\_, X, \_).\ \mathit{DetBy}(X, \mathsf{caller})$
	violation	$\mathit{some}\ \mathsf{sstore}(L_1, X, \_).\ \neg \mathit{MayDepOn}(X, \mathsf{caller}) \land \neg \mathit{MayDepOn}(L_1, \mathsf{caller})$
RT: Restricted transfer	*compliance*	$\mathit{all}\ \mathsf{call}(\_, \_, \_, \mathit{Amount}).\ \mathit{Amount} = 0$
	violation	$\mathit{some}\ \mathsf{call}(L_1, \_, \_, \mathit{Amount}).\ \mathit{DetBy}(\mathit{Amount}, \mathsf{data}) \land \neg \mathit{MayDepOn}(L_1, \mathsf{caller}) \land \neg \mathit{MayDepOn}(L_1, \mathsf{data})$
HE: Handled exception	*compliance*	$\mathit{all}\ \mathsf{call}(L_1, Y, \_, \_).\ \mathit{some}\ \mathsf{goto}(L_2, X, \_).\ \mathit{MustFollow}(L_1, L_2) \land \mathit{DetBy}(X, Y)$
	violation	$\mathit{some}\ \mathsf{call}(L_1, Y, \_, \_).\ \mathit{all}\ \mathsf{goto}(L_2, X, \_).\ \mathit{MayFollow}(L_1, L_2) \Rightarrow \neg \mathit{MayDepOn}(X, Y)$
TOD: Transaction ordering dependency	*compliance*	$\mathit{all}\ \mathsf{call}(\_, \_, \_, \mathit{Amount}).\ \neg \mathit{MayDepOn}(\mathit{Amount}, \mathsf{sload}) \land \neg \mathit{MayDepOn}(\mathit{Amount}, \mathsf{balance})$
	violation	$\mathit{some}\ \mathsf{call}(\_, \_, \_, \mathit{Amount}).\ \mathit{some}\ \mathsf{sload}(\_, Y, X).\ \mathit{some}\ \mathsf{sstore}(\_, X, \_).\ \mathit{DetBy}(\mathit{Amount}, Y) \land \mathit{isConst}(X)$
VA: Validated arguments	*compliance*	$\mathit{all}\ \mathsf{sstore}(L_1, \_, X).\ \mathit{MayDepOn}(X, \mathsf{arg}) \Rightarrow (\mathit{some}\ \mathsf{goto}(L_2, Y, \_).\ \mathit{MustFollow}(L_2, L_1) \land \mathit{DetBy}(Y, \mathsf{arg}))$
	violation	$\mathit{some}\ \mathsf{sstore}(L_1, \_, X).\ \mathit{DetBy}(X, \mathsf{arg}) \Rightarrow \neg(\mathit{some}\ \mathsf{goto}(L_2, Y, \_).\ \mathit{MayFollow}(L_2, L_1) \land \mathit{MayDepOn}(Y, \mathsf{arg}))$

Evaluation

1. Is Securify precise for relevant security properties?
2. How does Securify compare to other contract checkers?

How precise is Securify?

Dataset

- First 100 real-world contracts uploaded to https://securify.ch in 2018

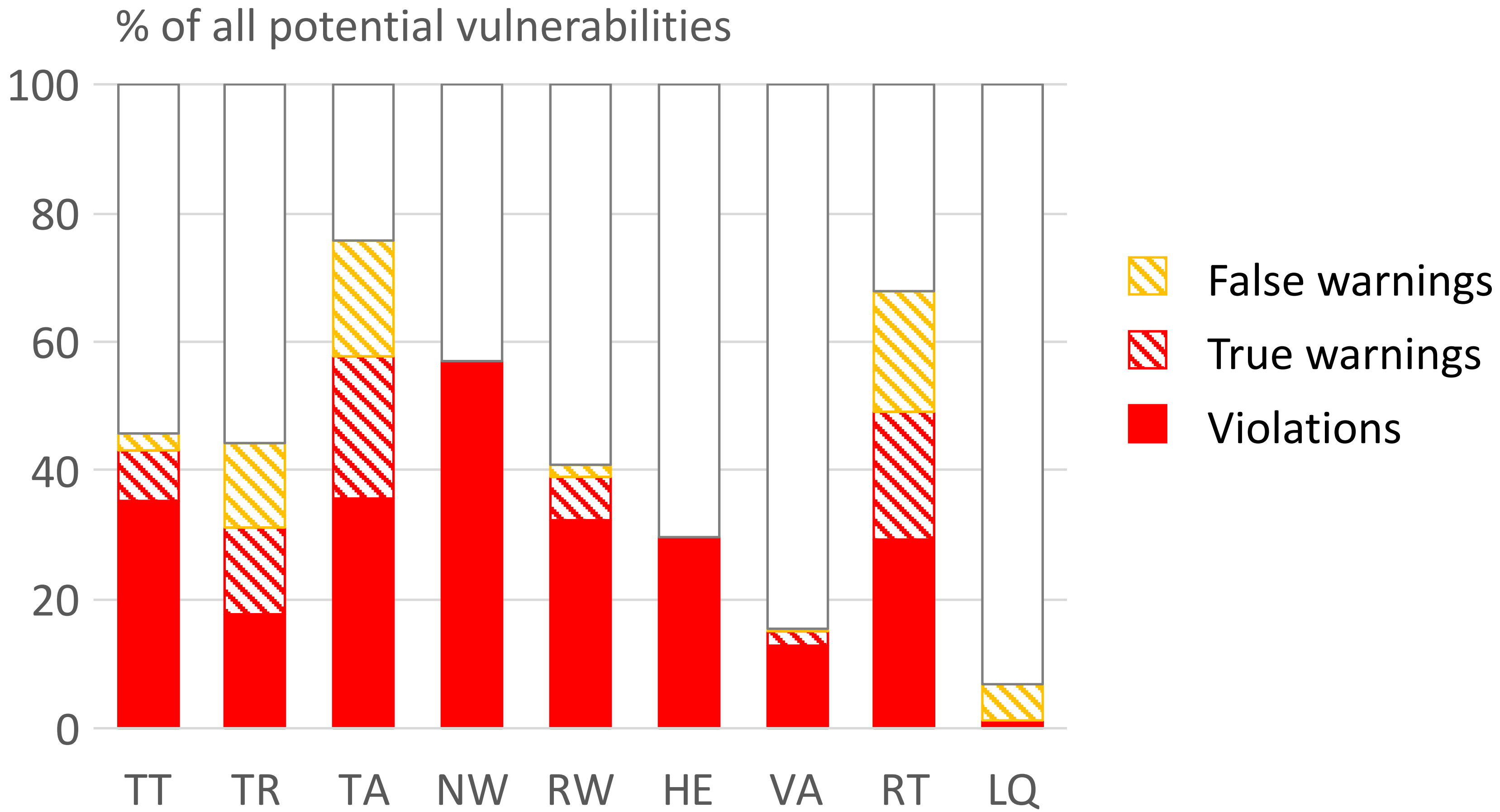
Security properties

- 9 critical vulnerabilities (reentrancy, …)

Experiment:

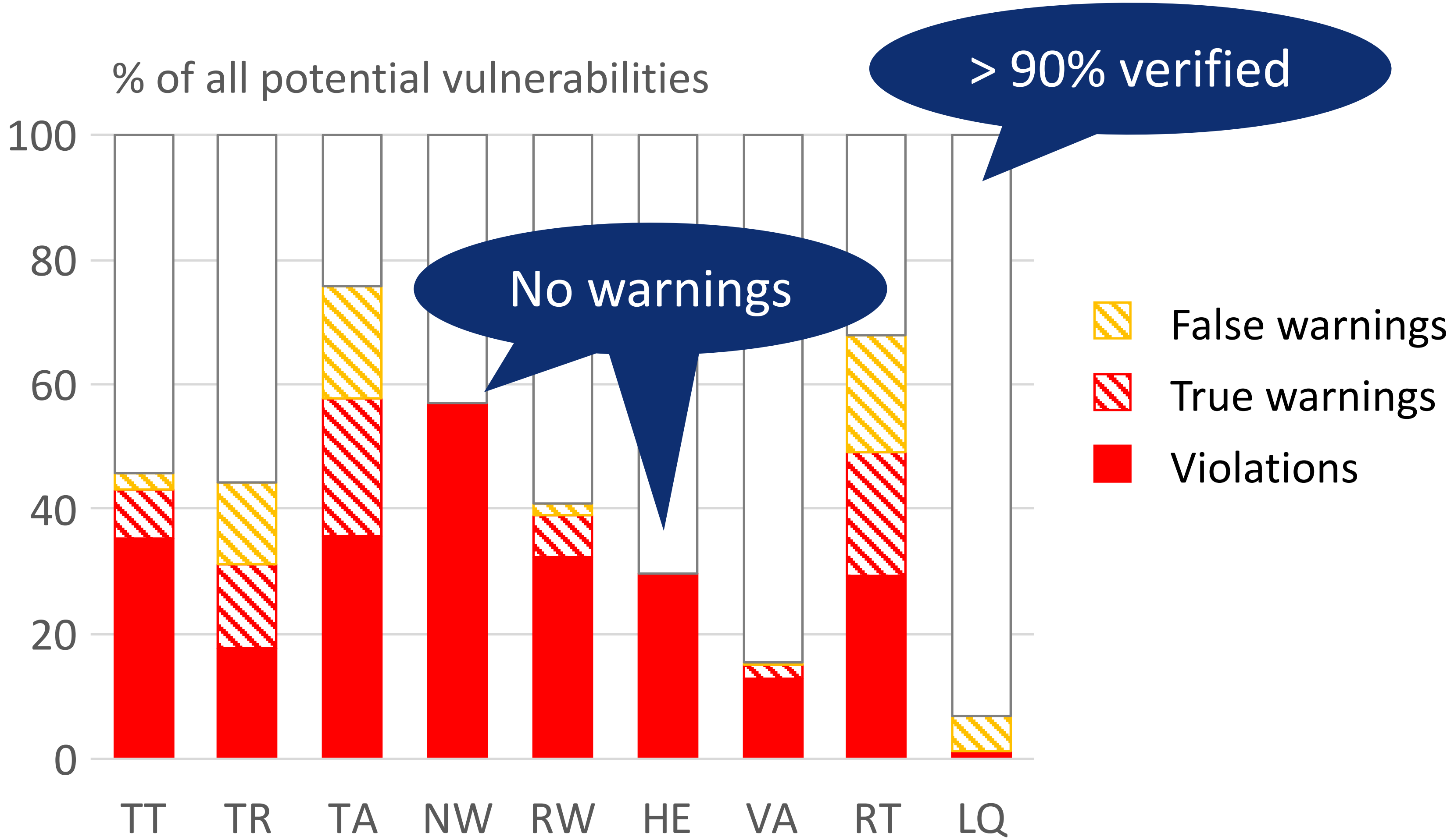
- Measure % of **violations**, **safe behaviors**, and **warnings**
- Manually classify **warnings** into **true warnings** and **false warnings**

How precise is Securify?



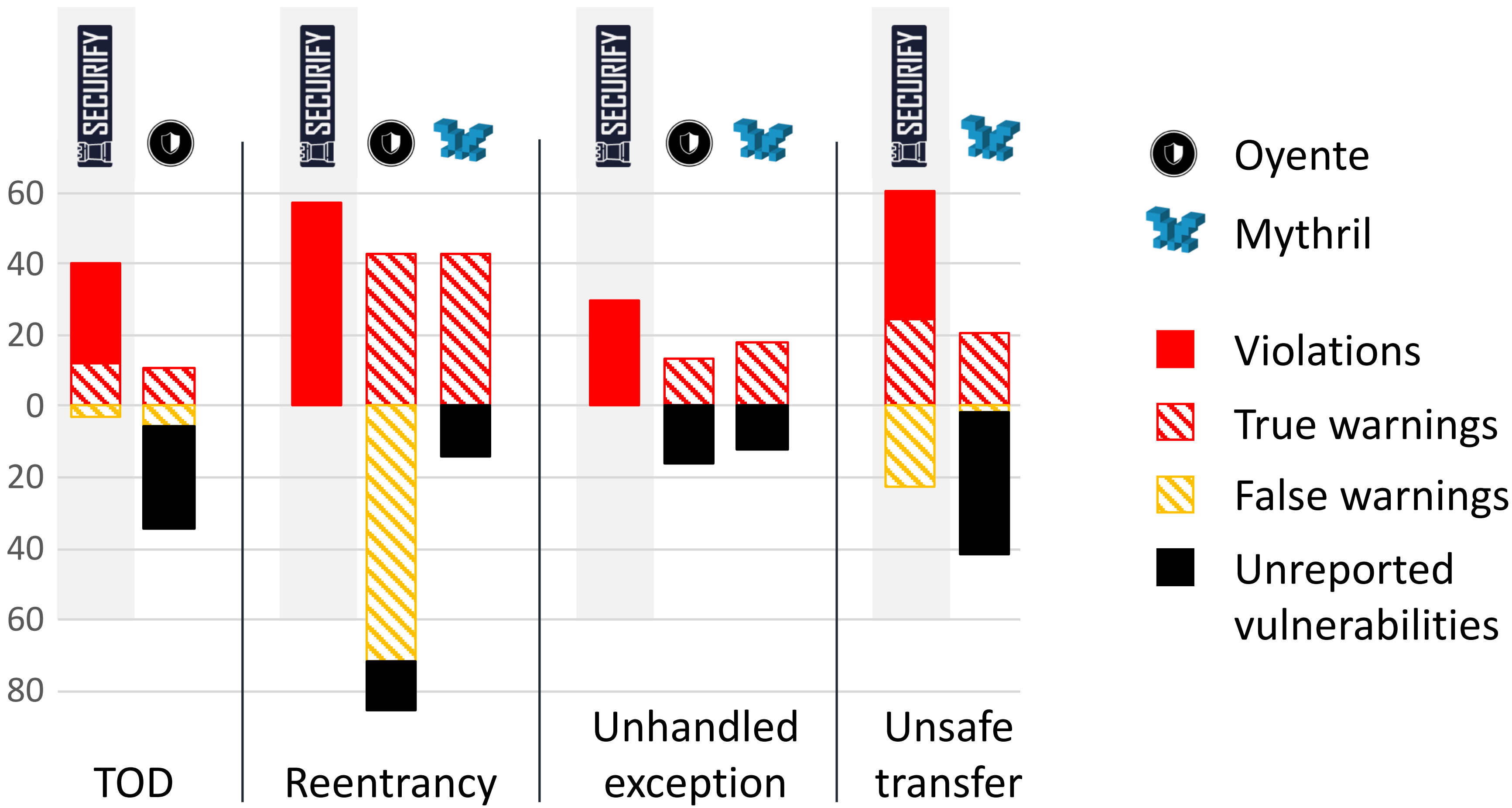
< 10% warnings for 6 out of 9 security properties

How precise is Securify?

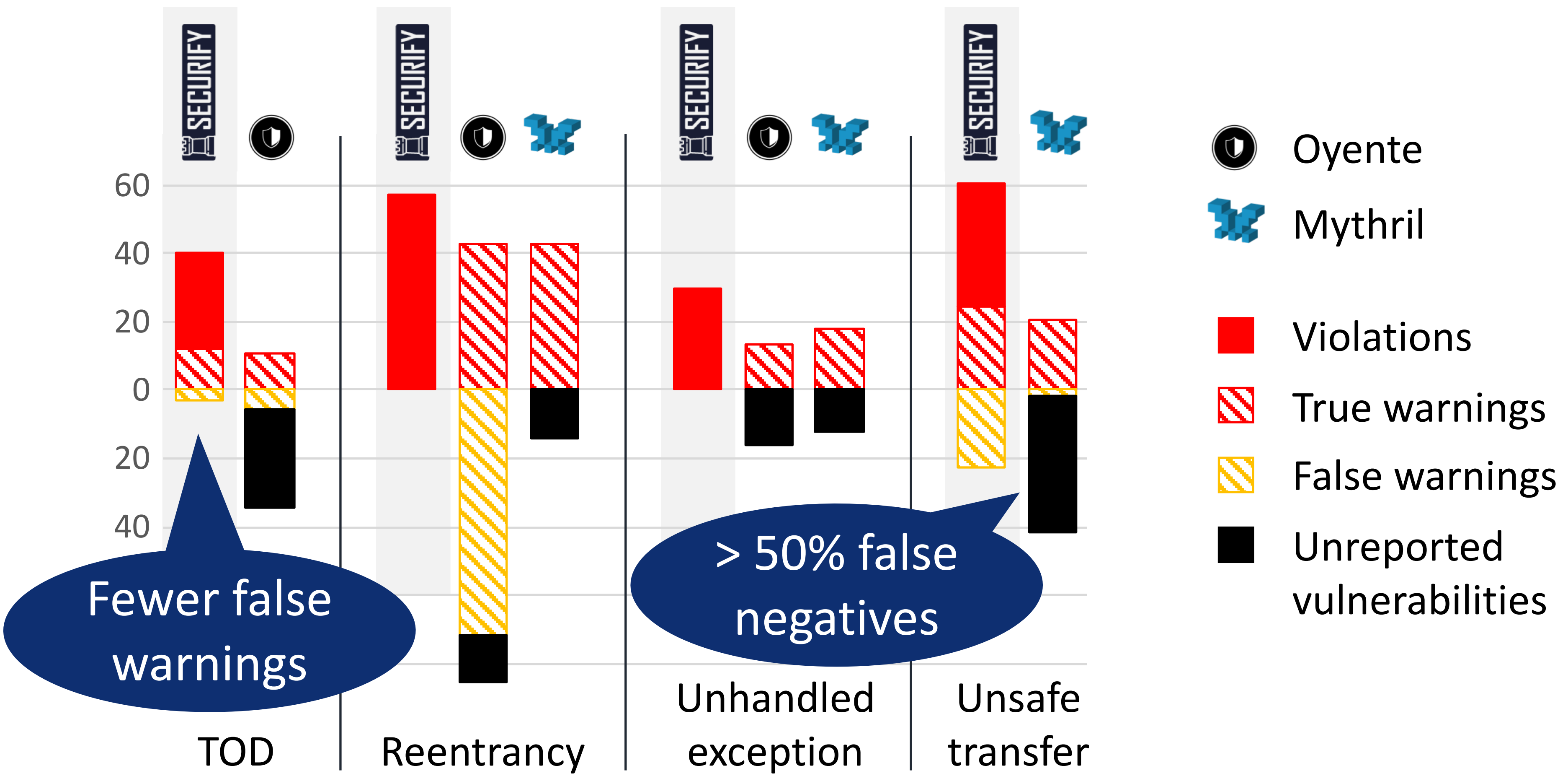


< 10% warnings for 6 out of 9 security properties

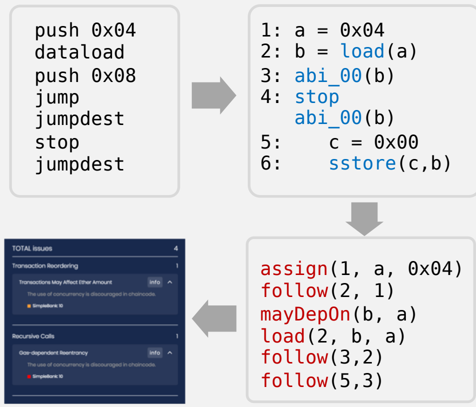
How does Securify compare to other checkers?



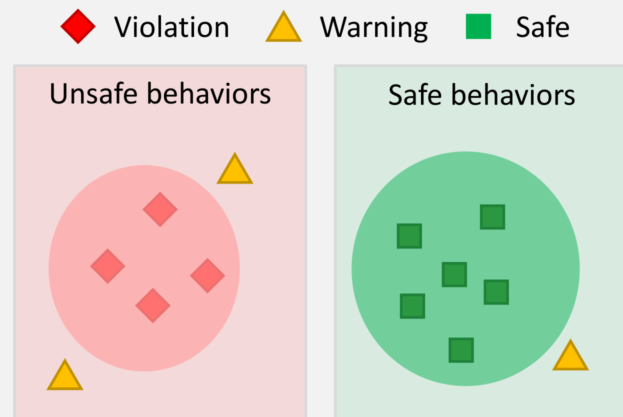
How does Securify compare to other checkers?



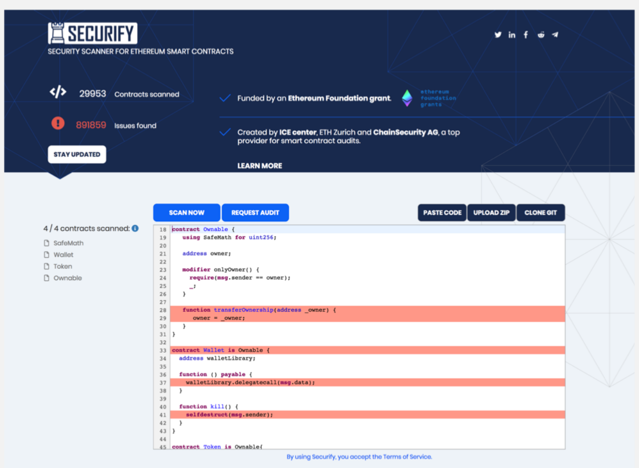
Summary



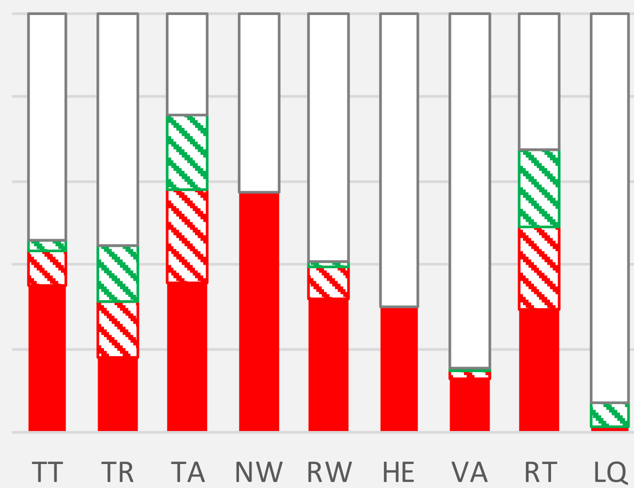
Scalable automated analysis



Domain-specific patterns



Try online: https://securify.ch



High precision on real contracts