Security analysis of smart contracts in Datalog

https://securify.ch

Dr. Petar Tsankov
Senior researcher, SRI lab, ETH Zurich
Co-founder and Chief Scientist, ChainSecurity

SRILAB

Inter-disciplinary research at ETH Zurich

Blockchain security

Safety of AI

Security and privacy

CHAINSECURITY

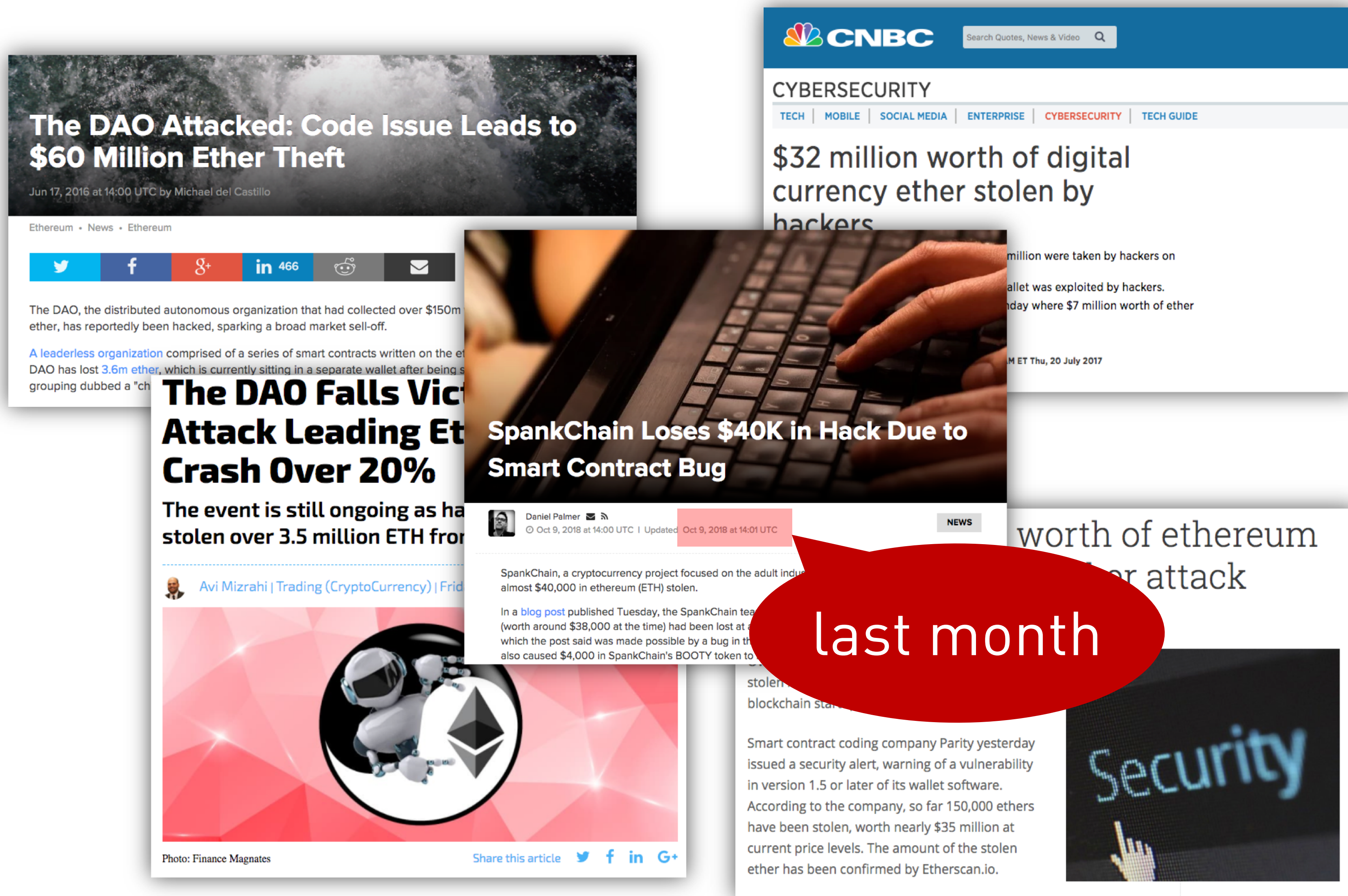
Next-generation blockchain security using automated reasoning

https://chainsecurity.com

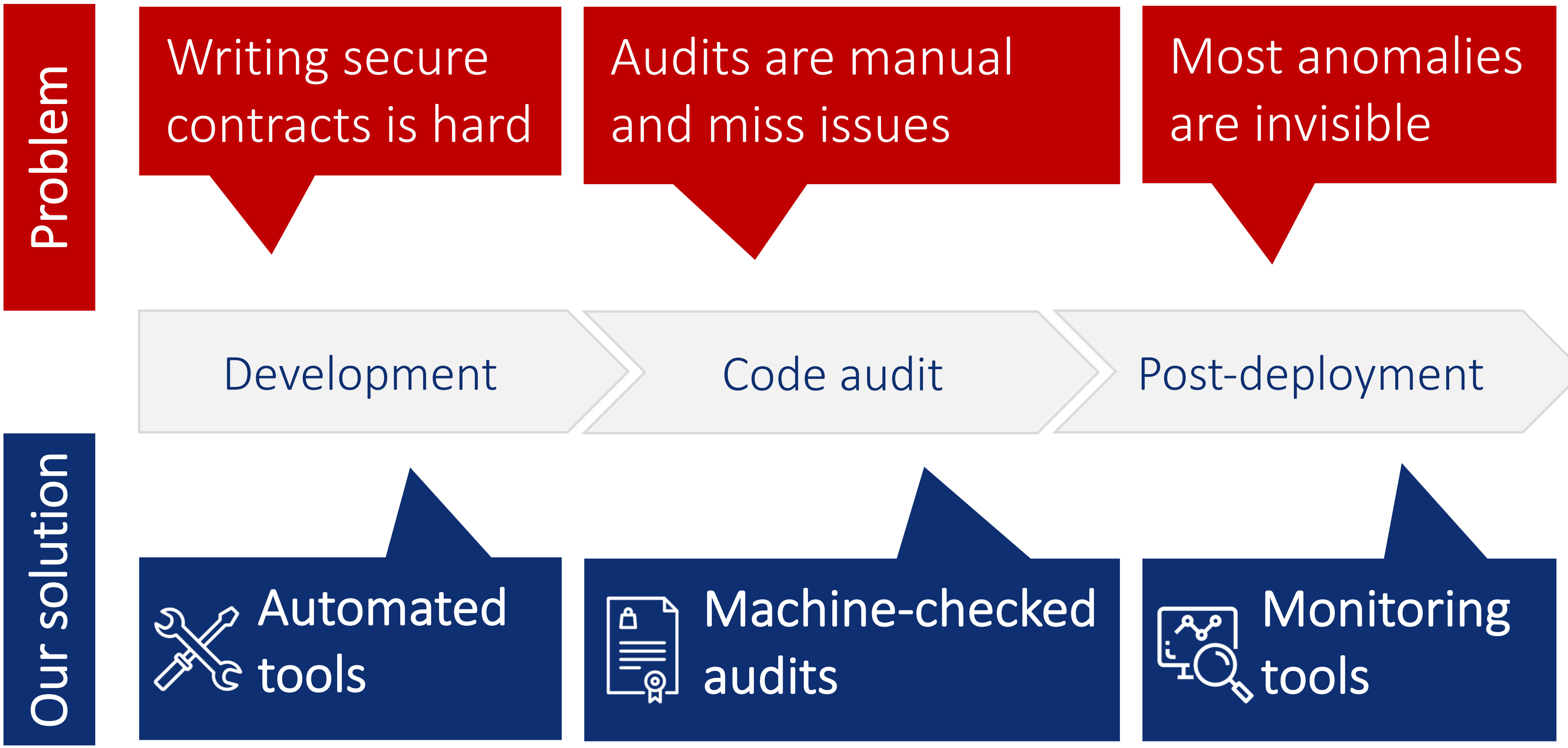
@chain_security

Why do we need reliable smart contracts?

Smart contract *bugs* in the news



Vision to secure smart contracts



Our core technology

SECURITY SCANNER

- Discovers *generic vulnerabilities*
- Supports Ethereum and Hyperledger

SYMBOLIC VERIFIER

- Supports *custom properties*
- Certifies correctness

AI-BASED TESTING

- Generates *high coverage* tests
- Learns from data (contracts and transactions)

June 2016: The DAO hack

The DAO hack

User contract

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

DAO contract

```
address => uint) balanc
 withdraw() {
mount = balances[msg.sender]
nder.call.value(amount)();
es[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

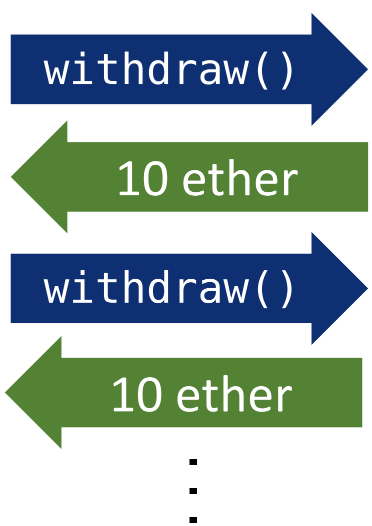
```
function foo() {
  dao.withdraw();
}
...
function () payable {
  dao.withdraw();
}
```

DAO 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



Many critical vulnerabilities

In 2017, more than

$300M

have been lost due to these issues

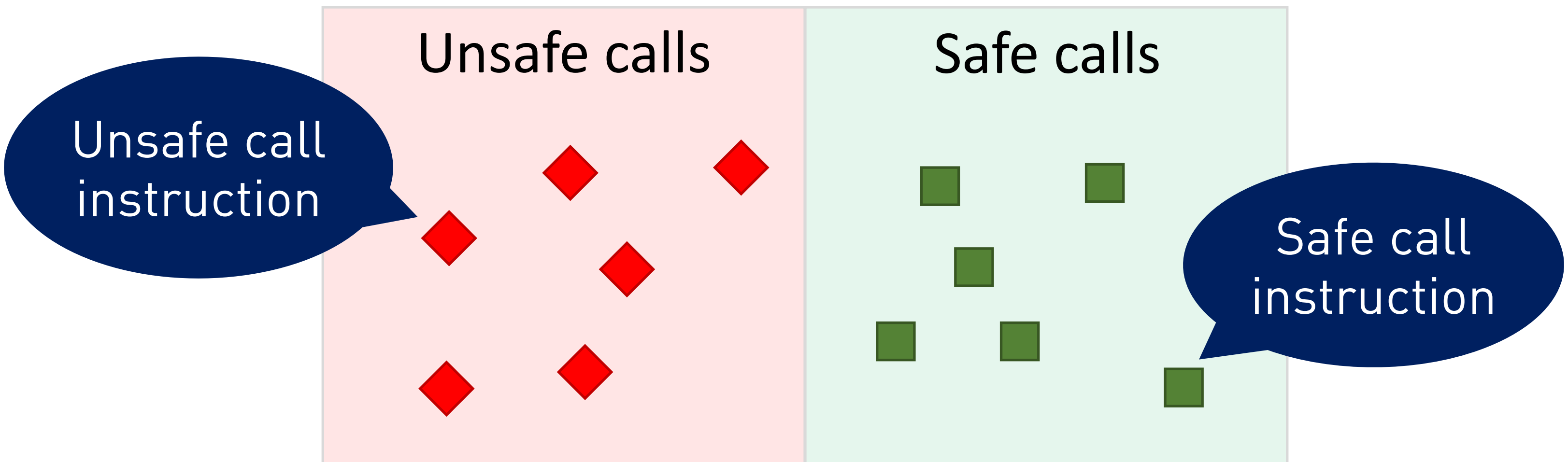
- Unexpected ether flows
- Unprivileged writes
- Use of unsafe inputs
- Reentrant method calls
- Transaction reordering

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?

No, smart contracts are Turing-complete

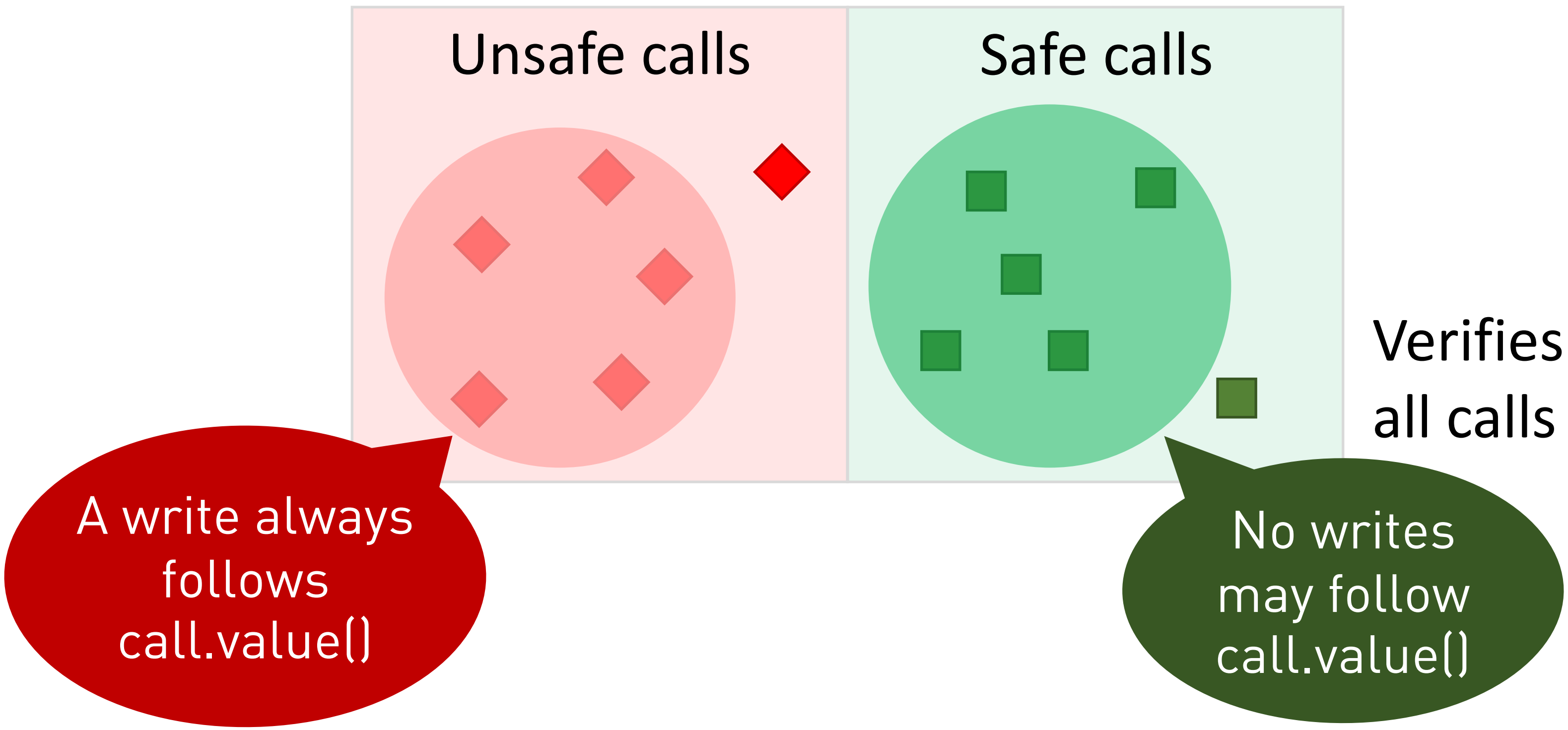
Insight

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

www.securify.ch

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

Impact

SECURIFY

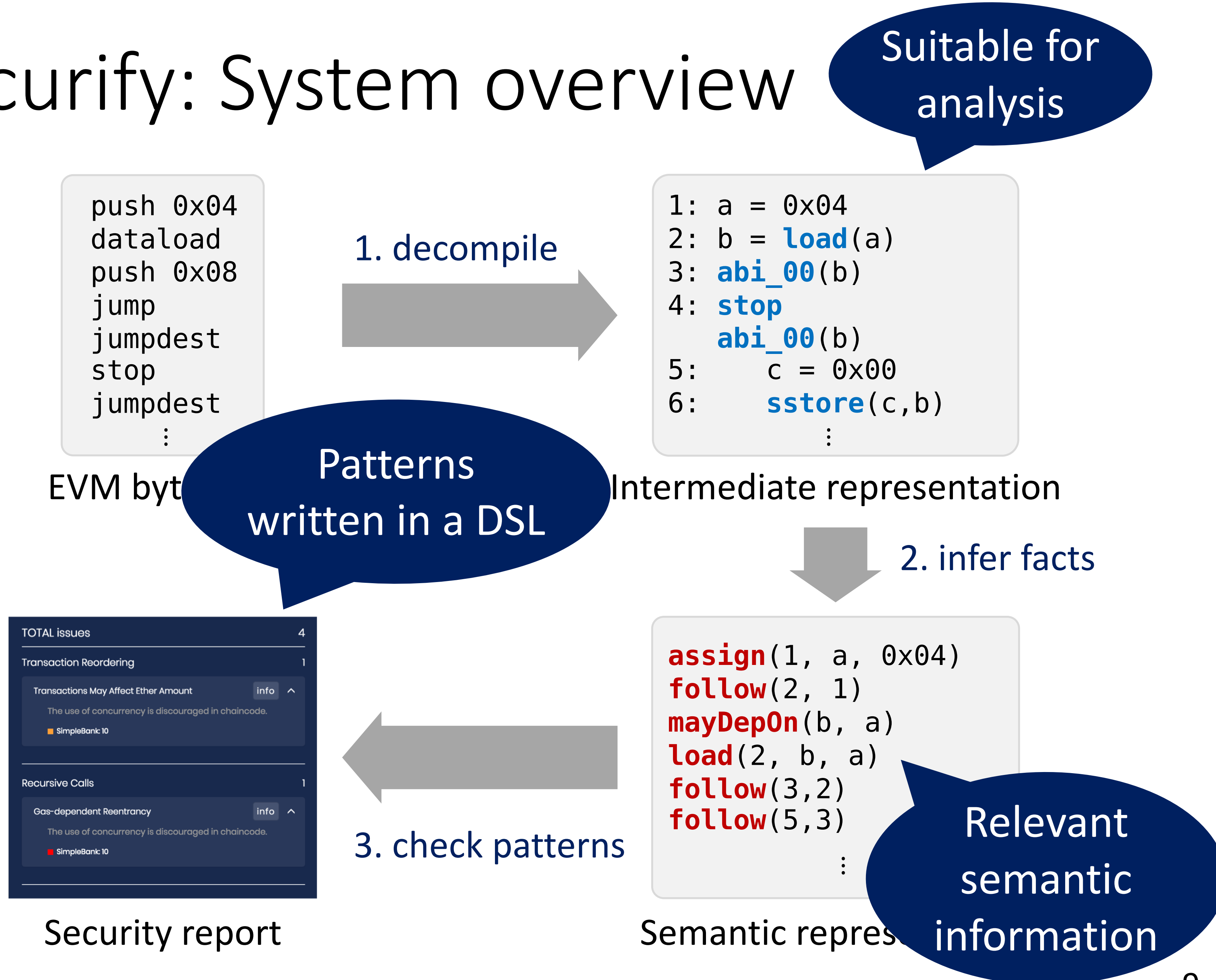
Used daily by security auditors
(30K+ contracts scanned so far)

1K+ subscribers

Grants: ethereum foundation

Startup: CHAINSECURITY

Securify: System overview



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

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

Datalog fixpoint

Step 2: Inferring semantic facts

Scalable inference of semantic facts using Datalog solvers

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

```
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

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

Datalog fixpoint

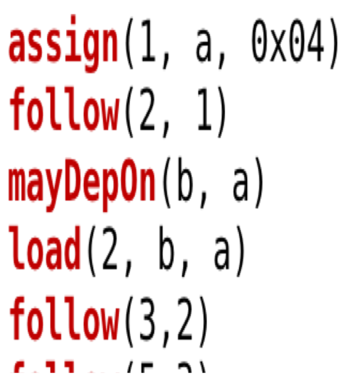
Step 2: Inferring semantic facts

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

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:

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

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;
}
```

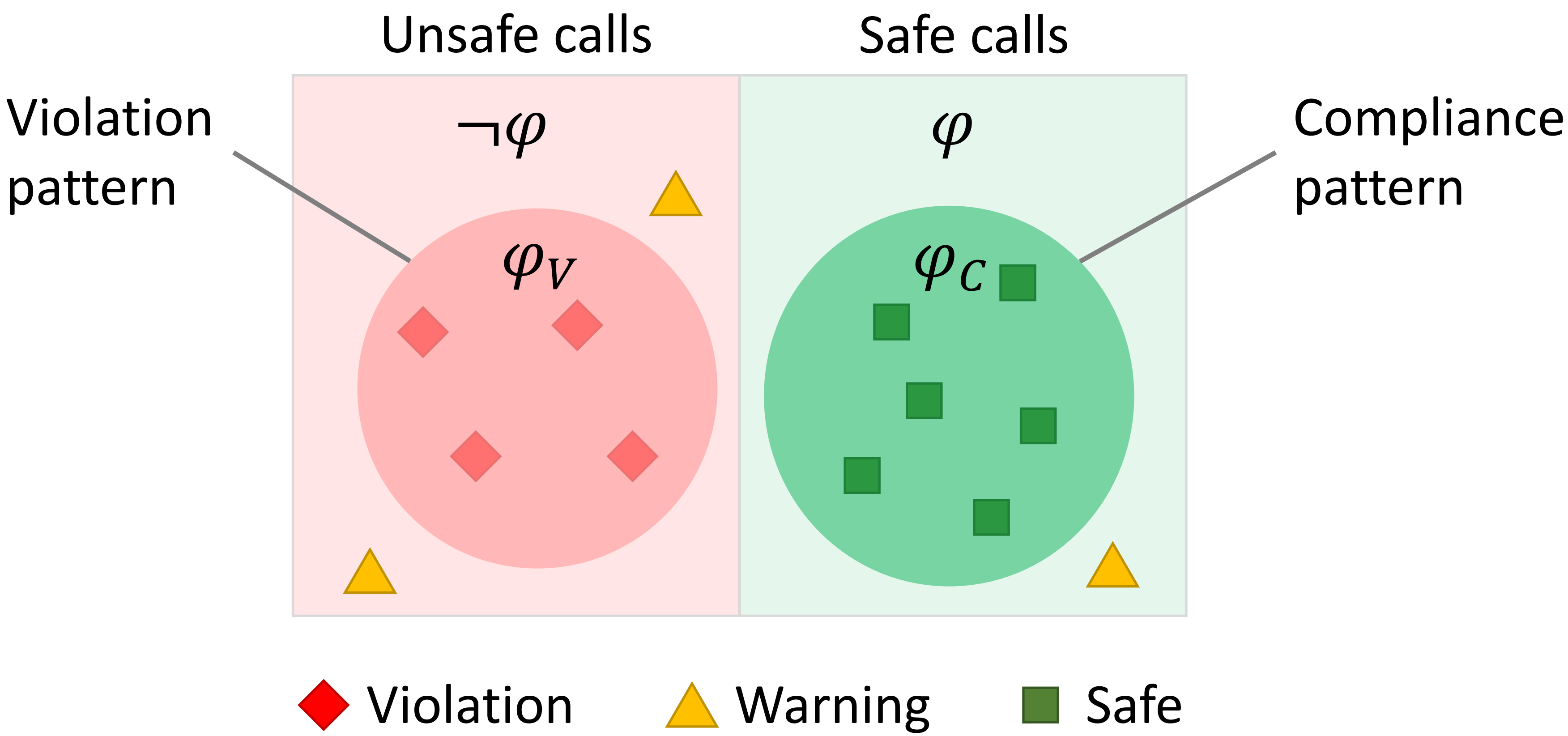
Security property: $\varphi \equiv$ "No state changes after call instructions"

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

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

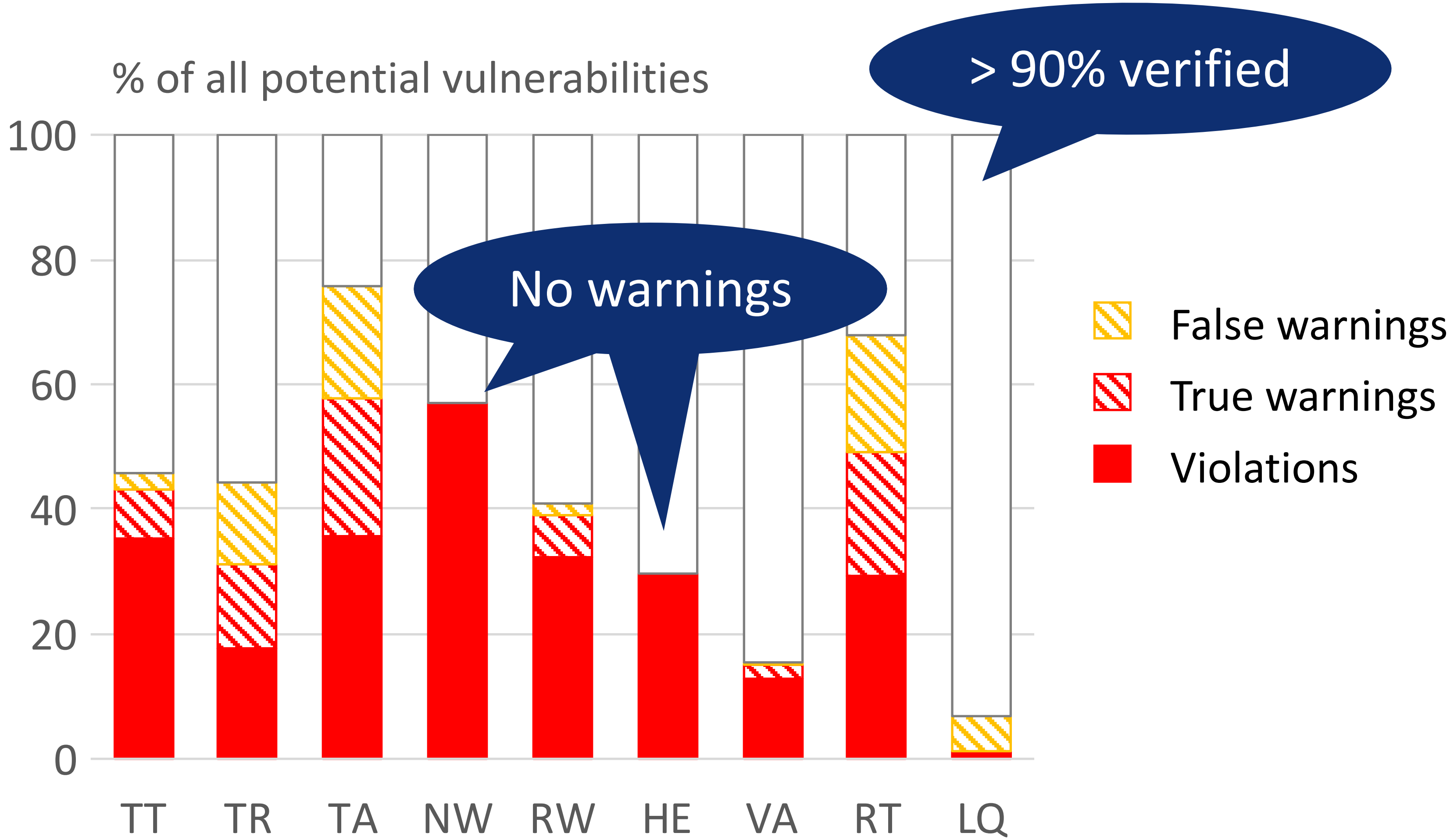
Security properties

- 9 critical vulnerabilities (reentrancy, …)

Experiment:

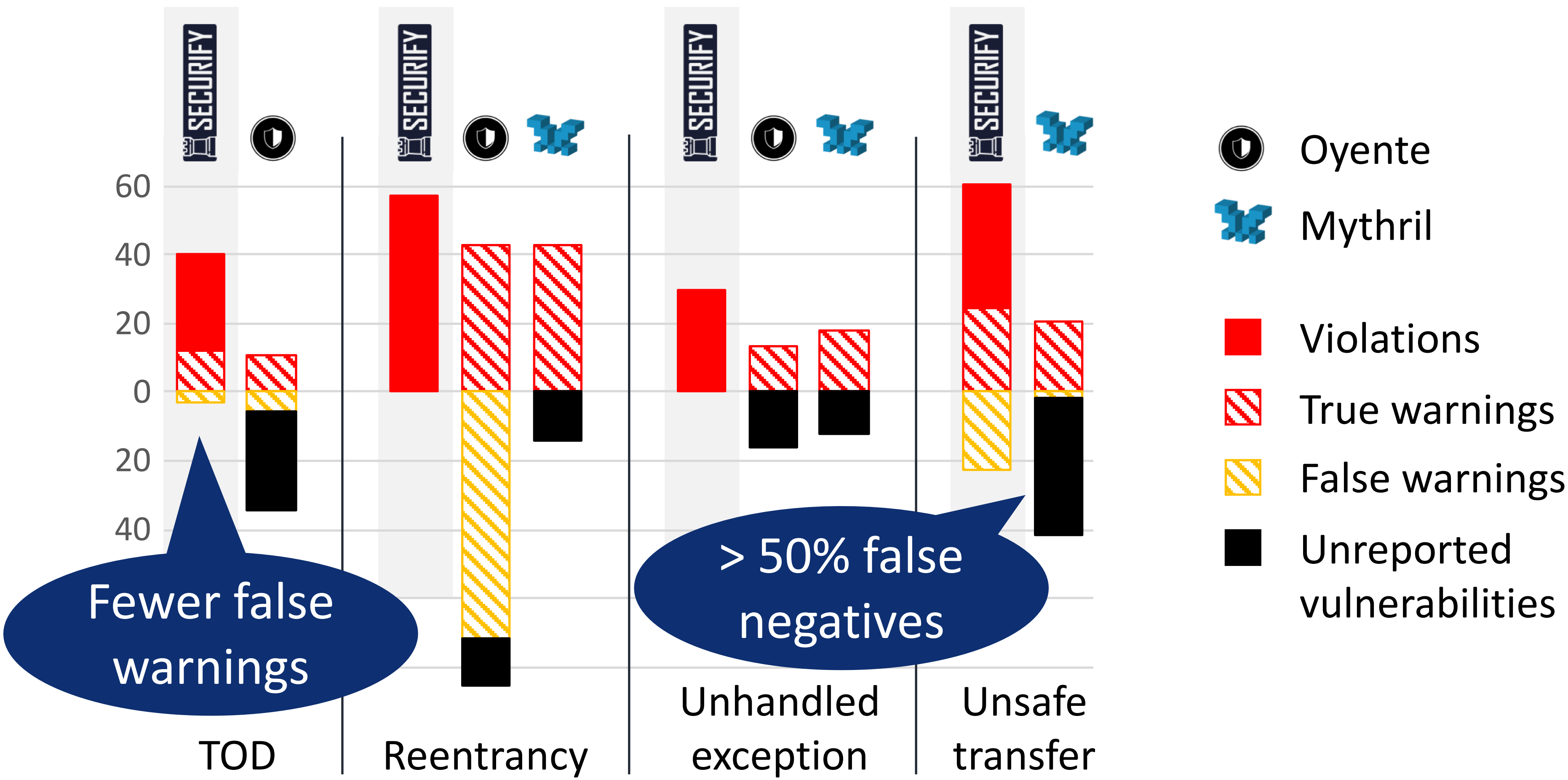
- Measure % of **violations**, **safe behaviors**, and **warnings**
- Manually classify **warnings** into **true warnings** and **false warnings**

How precise is Securify?

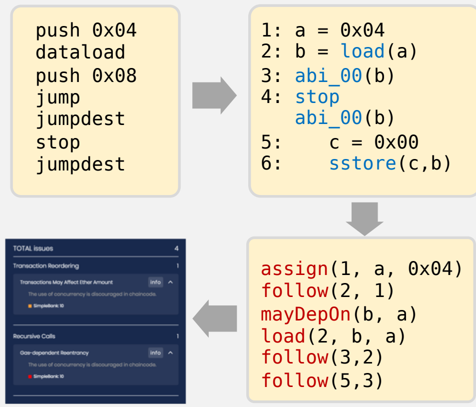


< 10% warnings for 6 out of 9 security properties

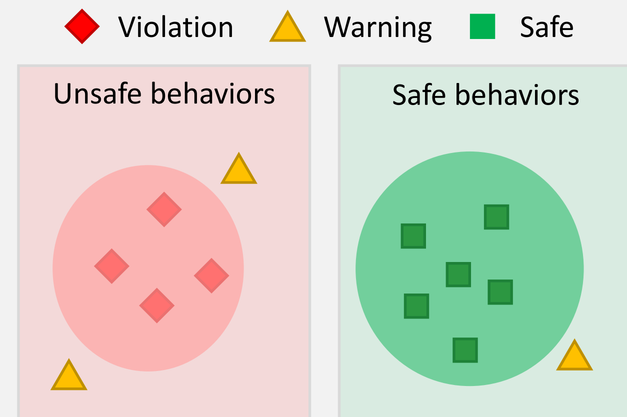
How does Securify compare to other checkers?



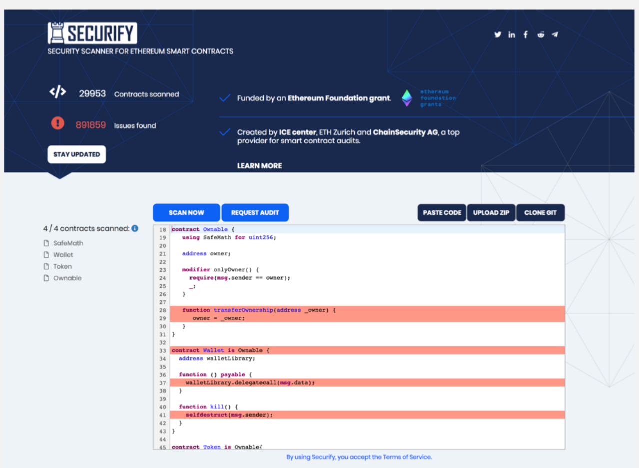
Summary



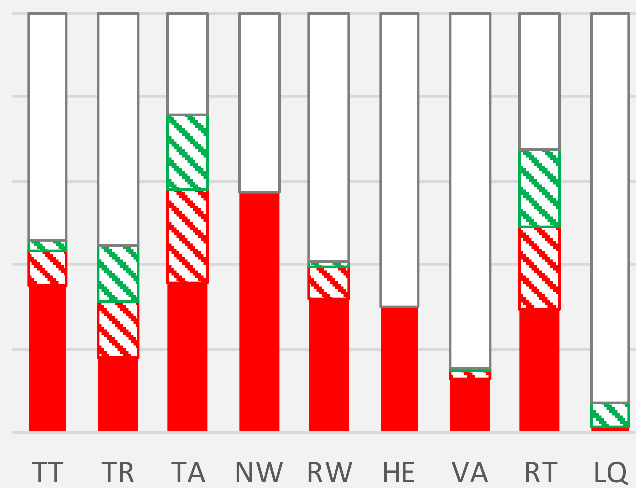
Scalable automated analysis



Precise security patterns



Try online: https://securify.ch



High precision on real contracts