



# 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



Inter-disciplinary research at  
ETH Zurich



Blockchain  
security



Safety of AI



Security  
and privacy



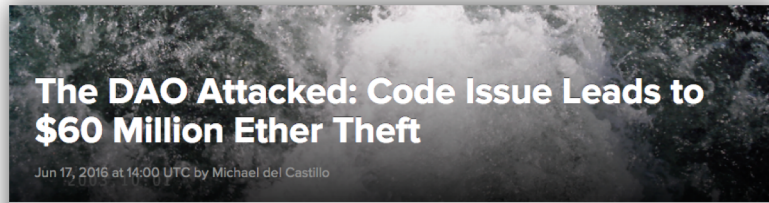
Next-generation blockchain security  
using automated reasoning

<https://chainsecurity.com>

@chain\_security

Why do we need reliable smart contracts?

# Smart contract *bugs* in the news



## The DAO Falls Victim to Attack Leading Ether Crash Over 20%

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



Avi Mizrahi | Trading (CryptoCurrency) | Friday

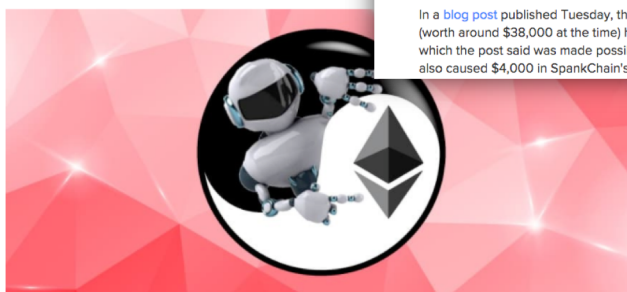


Photo: Finance Magnates

## SpankChain Loses \$40K in Hack Due to Smart Contract Bug



Daniel Palmer | Oct 9, 2018 at 14:00 UTC | Updated Oct 9, 2018 at 14:01 UTC

SpankChain, a cryptocurrency project focused on the adult industry, lost almost \$40,000 in ethereum (ETH) stolen.

In a [blog post](#) published Tuesday, the SpankChain team said that \$38,000 (worth around \$38,000 at the time) had been lost at the time, which the post said was made possible by a bug in the SpankChain smart contract, which also caused \$4,000 in SpankChain's BOOTY token to be lost.

last month

stolen  
blockchain stolen

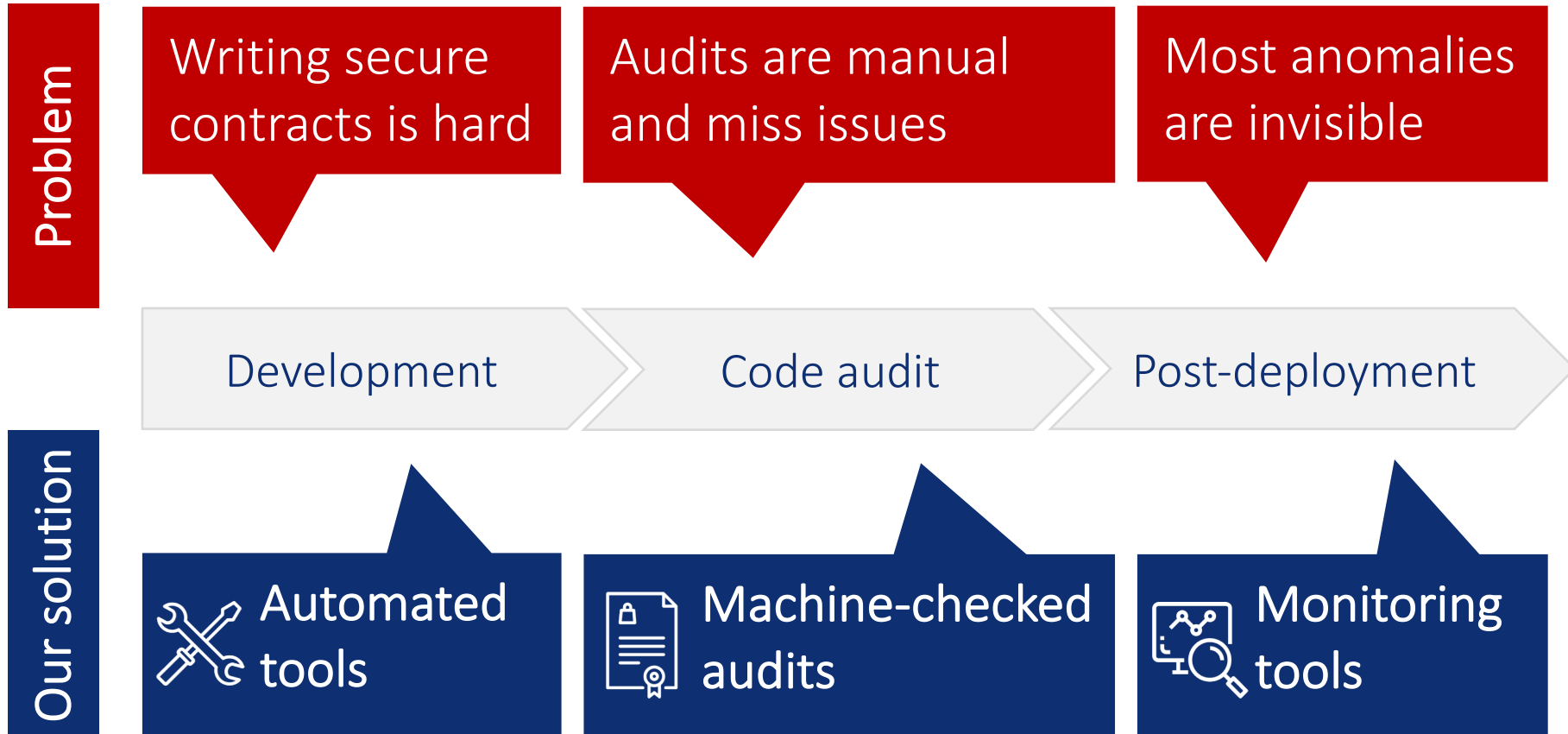
Smart contract coding company Parity yesterday issued a security alert, warning of a vulnerability in version 1.5 or later of its wallet software. According to the company, so far 150,000 ethers have been stolen, worth nearly \$35 million at current price levels. The amount of the stolen ether has been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

worth of ethereum  
attack





# 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



DAO contract

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

```
address => uint) balance  
  
withdraw() {  
  amount = balances[msg.sender]  
  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 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



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



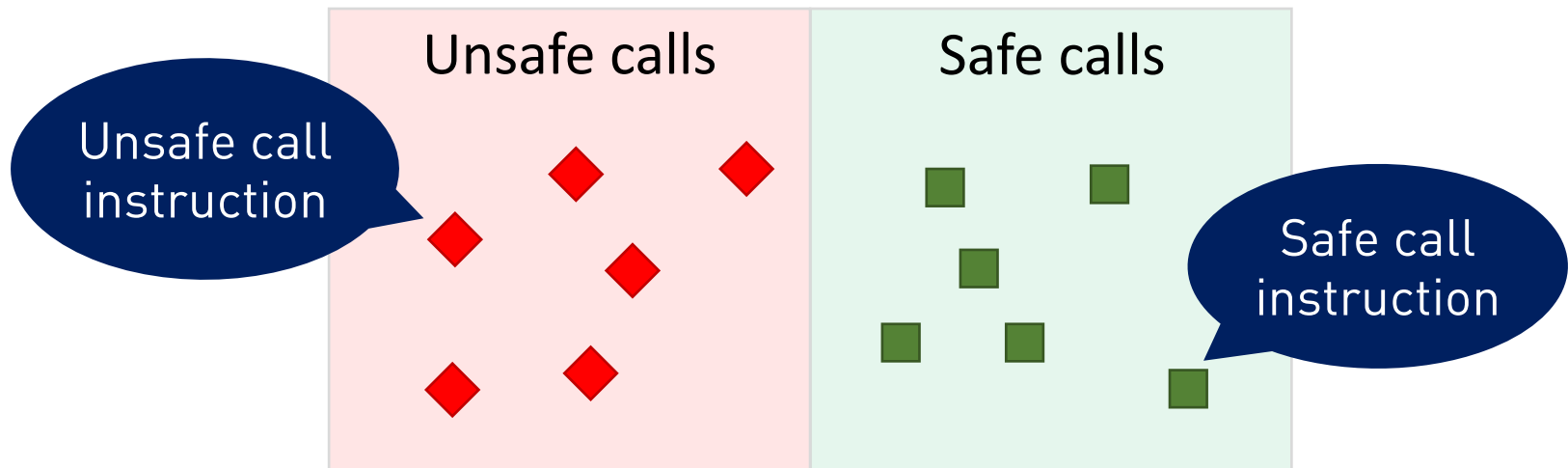
Transaction reordering

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?**

**No**, smart contracts are Turing-complete



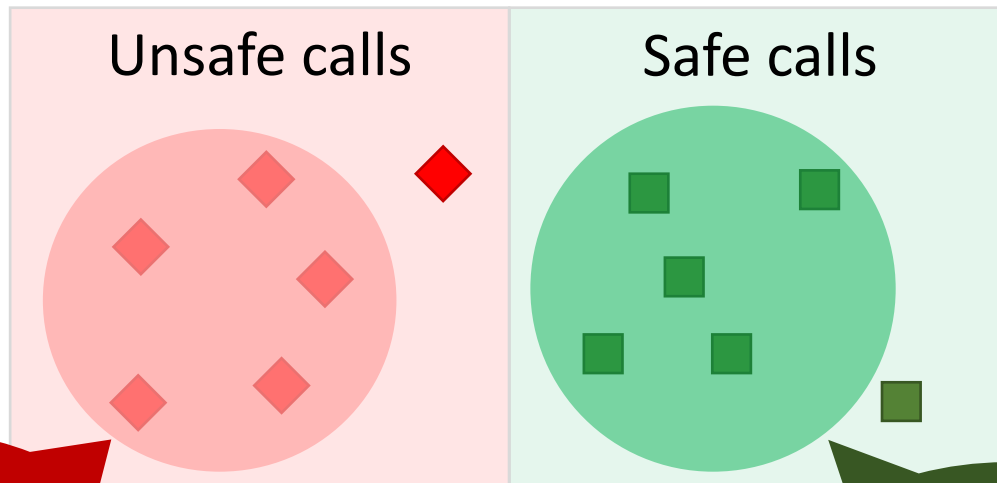
## Insight

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

A write always  
follows  
call.value()

Violation pattern

No writes  
may follow  
call.value()

Compliance pattern





SECURITY SCANNER FOR ETHEREUM SMART CONTRACTS



29953 Contracts scanned



Funded by an **Ethereum Foundation** grant.



ethereum  
foundation  
grants



89185



# SECURIFY

ETH Zurich and **ChainSecurity AG**, a top  
contract audits.

STAY UPDATED

SCAN NOW

REQUEST AUDIT

PASTE CODE

UPLOAD ZIP

CLONE GIT

4 / 4 contracts scanned:

- SafeMath
- Wallet
- Token
- Ownable

[www.securify.ch](http://www.securify.ch)

Scalable and fully *automated verifier* for  
Ethereum smart contracts

```
18 contract Ownable {
19     address owner;
20
21     address owner;
22
23     modifier onlyOwner() {
24         require(msg.sender == owner);
25     }
26
27     function transferOwnership(address _owner) {
28         owner = _owner;
29     }
30
31
32
33 contract Wallet is Ownable {
34     address walletLibrary;
35
36     function () payable {
37         walletLibrary.delegatecall(msg.data);
38     }
39
40     function kill() {
41         selfdestruct(msg.sender);
42     }
43 }
44
45 contract Token is Ownable{
```

By using Securify, you accept the [Terms of Service](#).

# Impact



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



1K+ subscribers

Grants:



Startup:



# Securify: System overview

Suitable for analysis

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

EVM byte

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)
:
```

Relevant semantic information

Semantic representation

3. check patterns

The screenshot shows a security report with a dark theme. At the top, it says 'TOTAL issues 4'. Below that, 'Transaction Reordering' is listed with a count of 1. A section titled 'Transactions May Affect Ether Amount' includes a warning: 'The use of concurrency is discouraged in chaincode.' and a link to 'SimpleBank 10'. Another section, 'Recursive Calls', also has a count of 1 and a warning about concurrency, with a link to 'SimpleBank 10'.

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) ← Follow(i,j)  
MayFollow(i,j) ← 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

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

---

<i>mayFollow</i> ( $L_1, L_2$ )	Instruction at label $L_1$ may follow that at label $L_2$
<i>mustFollow</i> ( $L_1, L_2$ )	Instruction at label $L_1$ must follow that at label $L_2$

---

### Data-flow analysis

---

<i>mayDepOn</i> ( $X, T$ )	The value of $X$ may depend on tag $T$
<i>eq</i> ( $X, T$ )	The values of $X$ and $T$ are equal
<i>detBy</i> ( $X, T$ )	For different values of $T$ the value of $X$ is different

---

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

in                      Catalog input                      Catalog output

# Step 3: Check patterns

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

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:

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

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;  
}
```

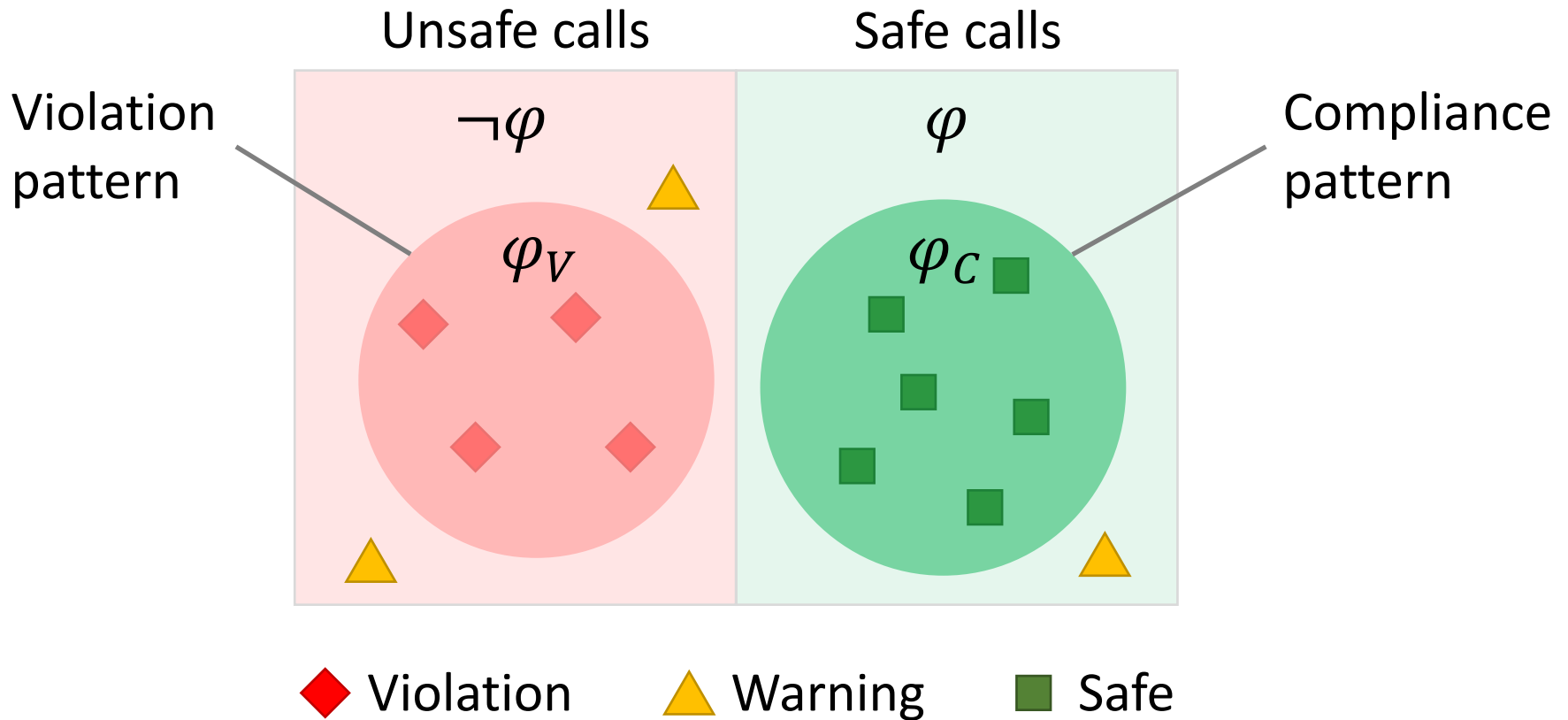
Security property:  $\varphi \equiv$  “No state changes after call instructions”

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

Violation pattern  $\varphi_V \equiv \exists \text{ call}(L_1, \_, \_). \exists \text{ sstore}(L_2, \_, \_). \text{mustFollow}(L_2, L_1)$

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
<b>LQ: Ether liquidity</b>	compliance	$\text{all stop}(L_1). \text{some goto}(L_2, X, L_3). X = \text{callvalue} \wedge \text{Follow}(L_2, L_4) \wedge L_3 \neq L_4 \wedge \text{MustFollow}(L_4, L_1)$
	compliance	$\text{some call}(L_1, \_, \_, \text{Amount}). \text{Amount} \neq 0 \vee \text{DetBy}(\text{Amount}, \text{data})$
	violation	$(\text{some stop}(L). \neg \text{MayDepOn}(L, \text{callvalue})) \wedge (\text{all call}(\_, \_, \_, \text{Amount}). \text{Amount} = 0)$
<b>NW: No writes after call</b>	compliance	$\text{all call}(L_1, \_, \_, \_). \text{all sstore}(L_2, \_, \_). \neg \text{MayFollow}(L_1, L_2)$
	violation	$\text{some call}(L_1, \_, \_, \_). \text{some sstore}(L_2, \_, \_). \text{MustFollow}(L_1, L_2)$
<b>RW: Restricted write</b>	compliance	$\text{all sstore}(\_, X, \_). \text{DetBy}(X, \text{caller})$
	violation	$\text{some sstore}(L_1, X, \_). \neg \text{MayDepOn}(X, \text{caller}) \wedge \neg \text{MayDepOn}(L_1, \text{caller})$
<b>RT: Restricted transfer</b>	compliance	$\text{all call}(\_, \_, \_, \text{Amount}). \text{Amount} = 0$
	violation	$\text{some call}(L_1, \_, \_, \text{Amount}). \text{DetBy}(\text{Amount}, \text{data}) \wedge \neg \text{MayDepOn}(L_1, \text{caller}) \wedge \neg \text{MayDepOn}(L_1, \text{data})$
<b>HE: Handled exception</b>	compliance	$\text{all call}(L_1, Y, \_, \_). \text{some goto}(L_2, X, \_). \text{MustFollow}(L_1, L_2) \wedge \text{DetBy}(X, Y)$
	violation	$\text{some call}(L_1, Y, \_, \_). \text{all goto}(L_2, X, \_). \text{MayFollow}(L_1, L_2) \Rightarrow \neg \text{MayDepOn}(X, Y)$
<b>TOD: Transaction ordering dependency</b>	compliance	$\text{all call}(\_, \_, \_, \text{Amount}). \neg \text{MayDepOn}(\text{Amount}, \text{load}) \wedge \neg \text{MayDepOn}(\text{Amount}, \text{balance})$
	violation	$\text{some call}(\_, \_, \_, \text{Amount}). \text{some sload}(\_, Y, X). \text{some sstore}(\_, X, \_). \text{DetBy}(\text{Amount}, Y) \wedge \text{isConst}(X)$
<b>VA: Validated arguments</b>	compliance	$\text{all sstore}(L_1, \_, X). \text{MayDepOn}(X, \text{arg})$
		$\Rightarrow (\text{some goto}(L_2, Y, \_). \text{MustFollow}(L_2, L_1) \wedge \text{DetBy}(Y, \text{arg}))$
	violation	$\text{some sstore}(L_1, \_, X). \text{DetBy}(X, \text{arg})$
		$\Rightarrow \neg (\text{some goto}(L_2, Y, \_). \text{MayFollow}(L_2, L_1) \wedge \text{MayDepOn}(Y, \text{arg}))$

# 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

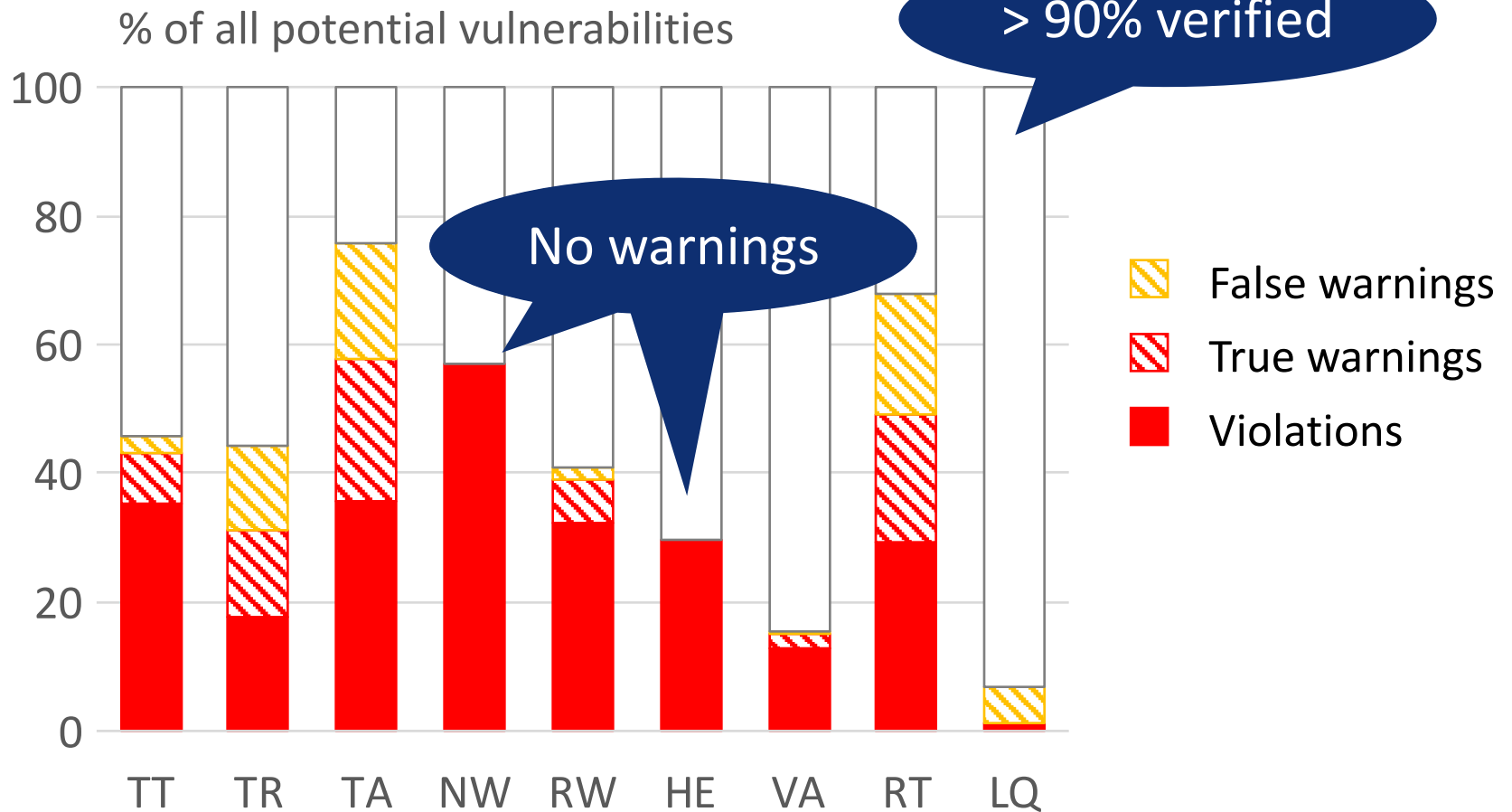
## Security properties

- 9 critical vulnerabilities (reentrancy, ...)

## Experiment:

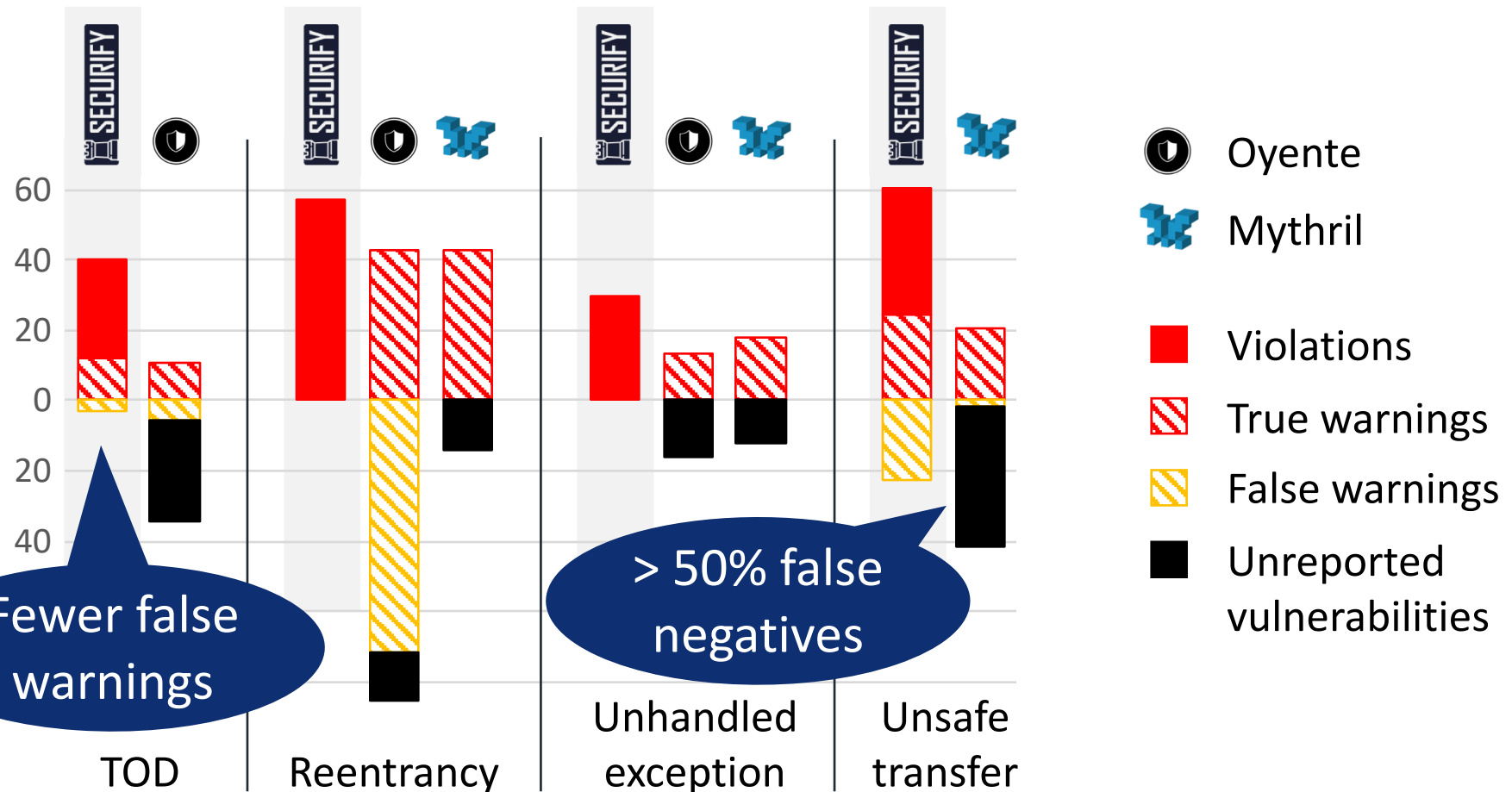
- Measure % of **violations**, **safe behaviors**, and **warnings**
- Manually classify **warnings** into **true warnings** and **false warnings**

# How precise is Securify?

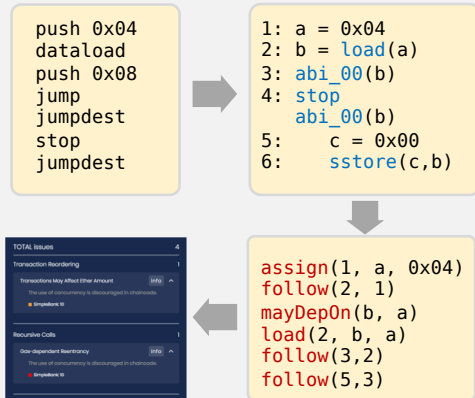


< 10% warnings for 6 out of 9 security properties

# How does Securify compare to other checkers?

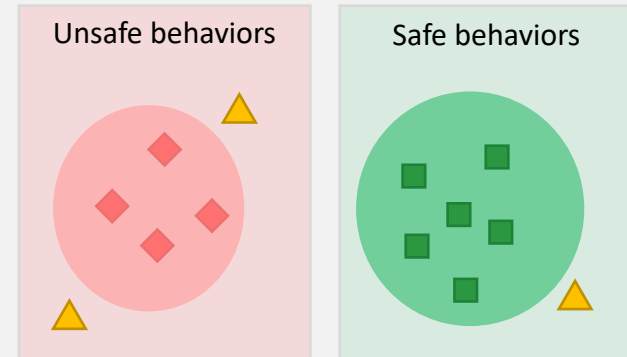


# Summary

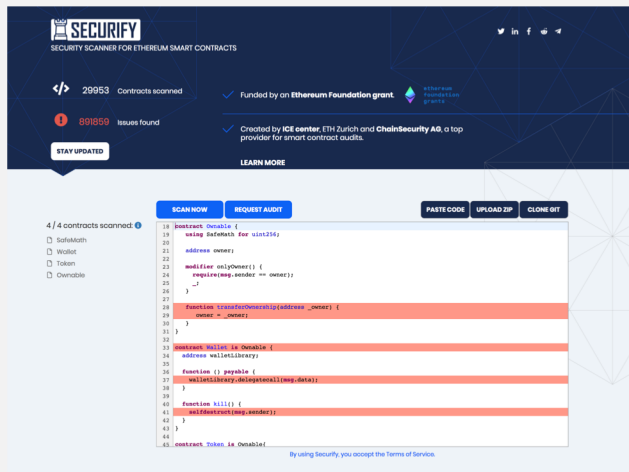


Scalable automated analysis

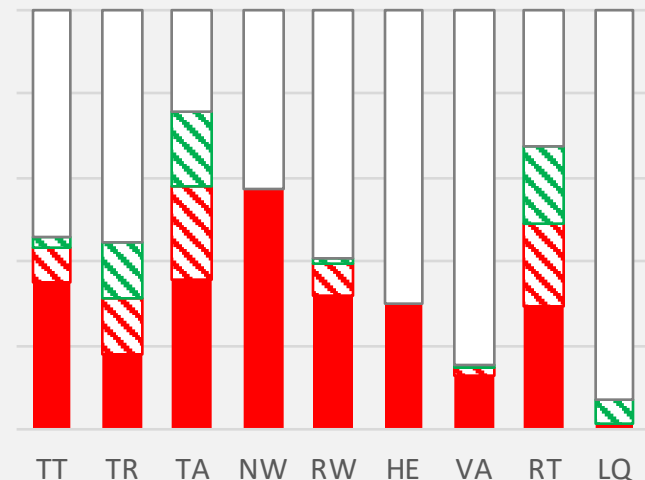
◆ Violation    ▲ Warning    ■ Safe



Precise security patterns



Try online: <https://securify.ch>



High precision on real contracts