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 last month.
Vision to secure smart contracts

**Problem**
- Writing secure contracts is hard
- Audits are manual and miss issues
- Most anomalies are invisible

**Our solution**
- Automated tools
- Machine-checked audits
- Monitoring tools

**Steps**
- Development
- Code audit
- Post-deployment
## Our core technology

<table>
<thead>
<tr>
<th>SECURITY SCANNER</th>
<th>SYMBOLIC VERIFIER</th>
<th>AI-BASED TESTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Discovers <em>generic vulnerabilities</em></td>
<td>- Supports <em>custom properties</em></td>
<td>- Generates <em>high coverage</em> tests</td>
</tr>
<tr>
<td>- Supports Ethereum and Hyperledger</td>
<td>- Certifies correctness</td>
<td>- Learns from data (contracts and transactions)</td>
</tr>
</tbody>
</table>
June 2016: The DAO hack
The DAO hack

User contract

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

DAO contract

```
address => uint) balances;

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

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

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

DAO contract

```solidity
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

Security property: No state changes after call instructions

Can we automatically find all unsafe calls?
No, smart contracts are Turing-complete
When contracts satisfy/violate a security property, they often satisfy/violate a simpler property.
The DAO hack

Security property: No state changes after call instructions

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

Unsafe calls

Safe calls

Verifies 91% of all calls

A write always follows call.value()

Violation pattern

No writes may follow call.value()

Compliance pattern
Scalable and fully *automated verifier* for Ethereum smart contracts
Impact

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

1K+ subscribers

Grants:  

- Ethereum Foundation

Startup:  

- ChainSecurity
Securify: System overview

1. decompile
   - EVM bytecode:
     ```
     push 0x04
data load
push 0x08
jump
jumpdest
stop
jumpdest
```

2. infer facts
   - Intermediate representation:
     ```
     1: a = 0x04
     2: b = load(a)
     3: abi_00(b)
     4: stop
        abi_00(b)
     5: c = 0x00
     6: sstore(c,b)
     ...```

3. check patterns
   - Semantic representation:
     ```
     assign(1, a, 0x04)
     follow(2, 1)
     mayDepOn(b, a)
     load(2, b, a)
     follow(3, 2)
     follow(5, 3)
     ...```

Suitable for analysis

Patterns written in a DSL

Security report

Relevant semantic information
Step 1: Decompilation

1. decompile

- Static single assignment form
- Control-flow graph recovery

EVM bytecode

push 0x04
dataload
push 0x08
jump
jumpdest
stop
jumpdest

Intermediate representation

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

...
Step 2: Inferring semantic facts

Intermediate representation

Semantic representation

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

IR

Datalog input

Datalog fixpoint
Step 2: Inferring semantic facts

Scalable inference of semantic facts using Datalog solvers

IR

Datalog input

Datalog fixpoint

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)

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

MayFollow(2, 1)
MayFollow(3, 1)
MayFollow(4, 1)
MayFollow(5, 1)
MayFollow(6, 1)
Step 2: Inferring semantic facts

Relevant semantic facts

<table>
<thead>
<tr>
<th>Control-flow analysis</th>
<th>Data-flow analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{mayFollow}(L_1, L_2)$</td>
<td>$\text{mayDepOn}(X, T)$</td>
</tr>
<tr>
<td>Instruction at label $L_1$ may follow that at label $L_2$</td>
<td>The value of $X$ may depend on tag $T$</td>
</tr>
<tr>
<td>$\text{mustFollow}(L_1, L_2)$</td>
<td>$\text{eq}(X, T)$</td>
</tr>
<tr>
<td>Instruction at label $L_1$ must follow that at label $L_2$</td>
<td>The values of $X$ and $T$ are equal</td>
</tr>
<tr>
<td></td>
<td>$\text{detBy}(X, T)$</td>
</tr>
<tr>
<td></td>
<td>For different values of $T$ the value of $X$ is different</td>
</tr>
</tbody>
</table>

For real-world contracts, Securify infers 1 - 10M such facts
Step 3: Check patterns

Security report

Semantic representation

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

```plaintext
assign(1, a, 0x04)
follow(2, 1)
mayDepOn(b, a)
load(2, b, a)
follow(3, 2)
follow(5, 3)
... (5)
```
Security patterns language

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

\[ \varphi ::= \text{instr}(L, Y, X, \ldots, X) \]

\[ \mid \text{eq}(X, T) \mid \text{detBy}(X, Y) \mid \text{mayDepOn}(X, Y) \]

\[ \mid \text{follow}(L, L) \mid \text{mayFollow}(L, L) \mid \text{mustFollow}(L, L) \]

\[ \mid \exists X. \varphi \mid \exists L. \varphi \mid \exists T. \varphi \mid \neg \varphi \mid \varphi \land \varphi \]

see paper for details
Example: No writes after calls

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

Security property: \( \varphi \equiv \text{“No state changes after call instructions”} \)

Compliance pattern: \( \varphi_c \equiv \forall \text{call}(L_1,\_,\_). \neg \exists \text{ssstore}(L_2,\_,\_). \text{mayFollow}(L_2,L_1) \)

Violation pattern: \( \varphi_c \equiv \exists \text{call}(L_1,\_,\_). \exists \text{ssstore}(L_2,\_,\_). \text{mustFollow}(L_2,L_1) \)

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Security Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQ: Ether liquidity violation</td>
<td>compliance</td>
<td>$(\forall \text{stop}(L_1). \ \text{some goto}(L_2, X, L_3). X = \text{callvalue} \land \text{Follow}(L_2, L_4) \land L_3 \neq L_4 \land \text{MustFollow}(L_4, L_1))$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some call}(L_1,<em>,</em>,\text{Amount}). \text{Amount} \neq 0 \lor \text{DetBy}(\text{Amount}, \text{data}))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(\text{some stop}(L). \neg \text{MayDepOn}(L, \text{callvalue})) \land (\text{all call}(<em>,</em>,_,\text{Amount}). \text{Amount} = 0)$</td>
</tr>
<tr>
<td>NW: No writes after call</td>
<td>compliance</td>
<td>$(\text{all call}(L_1,<em>,</em>,<strong>). \land \text{all sstore}(L_2,_,</strong>). \neg \text{MayFollow}(L_1, L_2))$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some sstore}(L_1,<em>,</em>,<strong>). \land \text{some sstore}(L_2,_,</strong>). \land \text{MustFollow}(L_1, L_2))$</td>
</tr>
<tr>
<td>RW: Restricted write violation</td>
<td>compliance</td>
<td>$(\text{all sstore}(_,X,__). \land \text{DetBy}(X, \text{caller}))$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some sstore}(L_1, X,__). \land \neg \text{MayDepOn}(X, \text{caller}) \land \neg \text{MayDepOn}(L_1, \text{caller}))$</td>
</tr>
<tr>
<td>RT: Restricted transfer</td>
<td>compliance</td>
<td>$(\text{all call}(<em>,</em>,_,\text{Amount}). \text{Amount} = 0)$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some call}(L_1,<em>,</em>,\text{Amount}). \land \text{DetBy}(\text{Amount}, \text{data}) \land \neg \text{MayDepOn}(L_1, \text{caller}) \land \neg \text{MayDepOn}(L_1, \text{data}))$</td>
</tr>
<tr>
<td>HE: Handled exception</td>
<td>compliance</td>
<td>$(\text{all call}(L_1, Y, _,__). \land \text{some goto}(L_2, X, _). \land \text{MustFollow}(L_1, L_2) \land \text{DetBy}(X, Y))$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some call}(L_1, Y, _,__). \land \text{all goto}(L_2, X, _). \land \text{MayDepOn}(L_1, L_2)) \Rightarrow \neg \text{MayDepOn}(X, Y))$</td>
</tr>
<tr>
<td>TOD: Transaction ordering dependency</td>
<td>compliance</td>
<td>$(\text{all call}(<em>,</em>,_,\text{Amount}). \land \neg \text{MayDepOn}(\text{Amount}, \text{sload}) \land \neg \text{MayDepOn}(\text{Amount}, \text{balance}))$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some call}(<em>,</em>,<em>,\text{Amount}). \land \text{some sload}(</em>, Y, X). \land \text{some sstore}(X, _,__). \land \text{DetBy}(\text{Amount}, Y) \land \text{isConst}(X))$</td>
</tr>
<tr>
<td>VA: Validated arguments</td>
<td>compliance</td>
<td>$(\text{all sstore}(L_1, _, X). \land \text{MayDepOn}(X, \text{arg})) \Rightarrow (\text{some goto}(L_2, Y, _). \land \text{MustFollow}(L_2, L_1) \land \text{DetBy}(Y, \text{arg}))$</td>
</tr>
<tr>
<td></td>
<td>violation</td>
<td>$(\text{some sstore}(L_1, _, X). \land \text{DetBy}(X, \text{arg})) \Rightarrow \neg (\text{some goto}(L_2, Y, _). \land \text{MayDepOn}(Y, \text{arg}))$</td>
</tr>
</tbody>
</table>
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?

% of all potential vulnerabilities

- TT
- TR
- TA
- NW
- RW
- HE
- VA
- RT
- LQ

- False warnings
- True warnings
- Violations

> 90% verified

No warnings

< 10% warnings for 6 out of 9 security properties
How does Securify compare to other checkers?

- Fewer false warnings
- > 50% false negatives

- TOD
- Reentrancy
- Unhandled exception
- Unsafe transfer

- Oyente
- Mythril

- Violations
- True warnings
- False warnings
- Unreported vulnerabilities
Summary

Scalable automated analysis

Precise security patterns

Try online: https://securify.ch

High precision on real contracts

push 0x04
data load
push 0x08
jump
jump dest
stop
jump dest

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

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

push 0x04
data load
push 0x08
jump
jump dest
stop
jump dest

Violation
Warning
Safe behaviors
Unsafe behaviors

Summary
High precision on real contracts
Scalable automated analysis
Precise security patterns
Unsafe behaviors
Safe behaviors
Violation
Warning
Safe

Try online: https://securify.ch