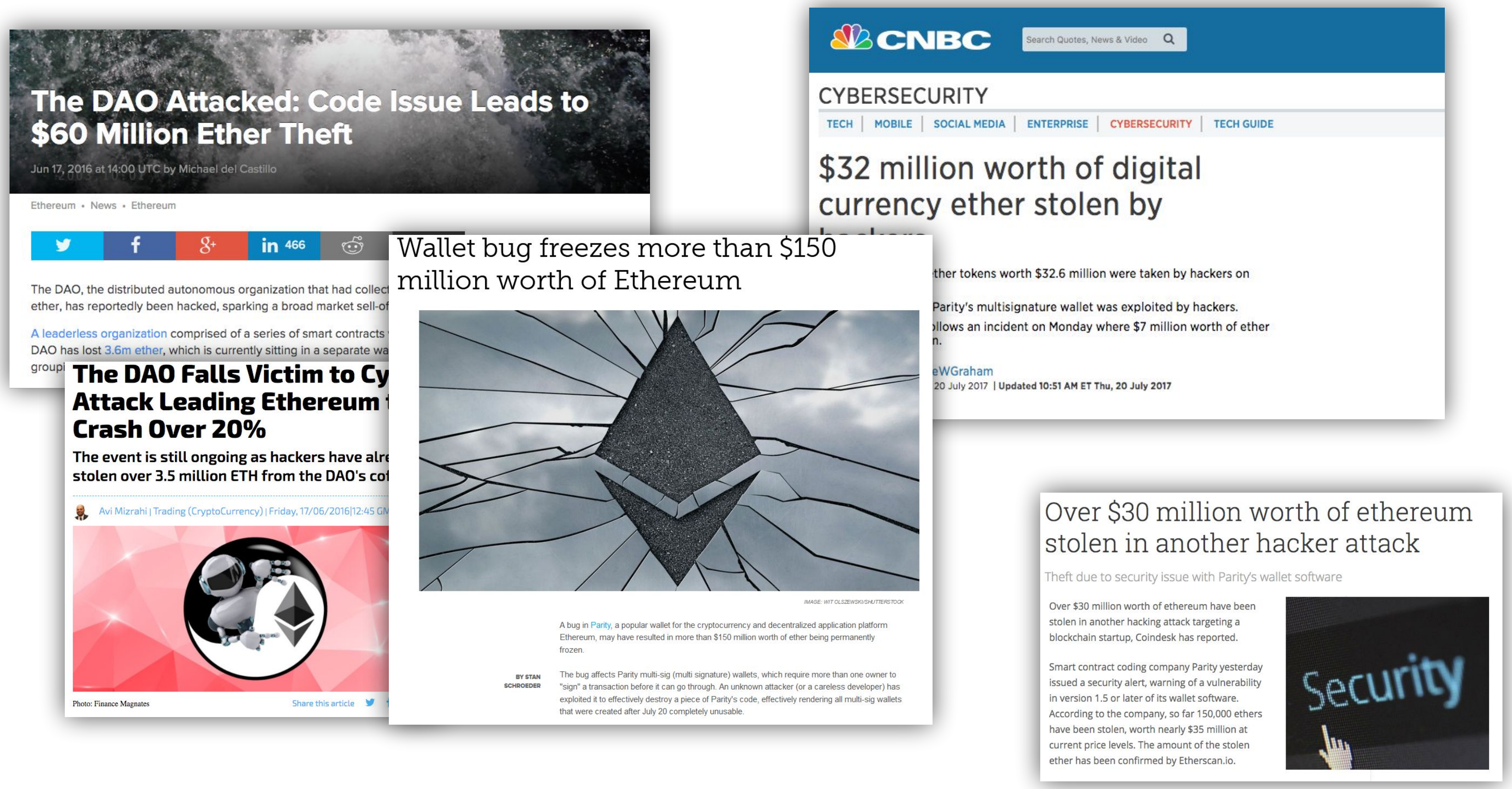
# Declarative Static Analysis of Smart Contracts

securify.ch

Quentin Hibon

Blockchain Security Engineer, ChainSecurity

# 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

The DAO, the distributed autonomous organization that had collec
ether, has reportedly been hacked, sparking a broad market sell-of

A leaderless organization comprised of a series of smart contracts
DAO has lost 3.6m ether, which is currently sitting in a separate wa
groupi

## The DAO Falls Victim to Cy
## Attack Leading Ethereum t
## Crash Over 20%

The event is still ongoing as hackers have alre
stolen over 3.5 million ETH from the DAO's co

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

Photo: Finance Magnates

Share this article

## Wallet bug freezes more than $150 million worth of Ethereum

IMAGE: WIT OLSZEWSKI/SHUTTERSTOCK

BY STAN SCHROEDER

A bug in Parity, a popular wallet for the cryptocurrency and decentralized application platform Ethereum, may have resulted in more than $150 million worth of ether being permanently frozen.

The bug affects Parity multi-sig (multi signature) wallets, which require more than one owner to "sign" a transaction before it can go through. An unknown attacker (or a careless developer) has exploited it to effectively destroy a piece of Parity's code, effectively rendering all multi-sig wallets that were created after July 20 completely unusable.

CYBERSECURITY

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

## $32 million worth of digital currency ether stolen by

ther tokens worth $32.6 million were taken by hackers on

Parity's multisignature wallet was exploited by hackers.

llows an incident on Monday where $7 million worth of ether
n.

eWGraham

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

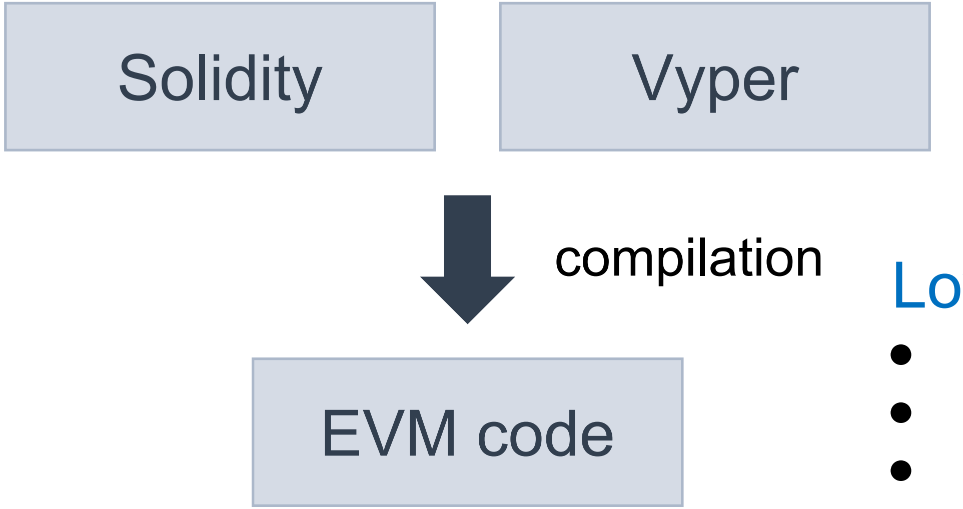
## Over $30 million worth of ethereum stolen in another hacker attack

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.

# Low-level Code



High-level languages

## Low-level code

- Stack-based
- Untyped
- No functions
- Not designed with formal analysis in mind

# Ethereum Virtual Machine (EVM)

| Operation type | Description | OPCodes |
|---|---|---|
| Arithmetic | Encode calculations | Add, Mul, Sub, Div, LT, EQ |
| Control-flow | Encode conditional jumps | Jump, JumpI |
| Cryptography | Compute hash functions | SHA3 |
| Environment | Fetch transaction information | Balance, Caller, Gas, Timestamp... |
| Memory / storage | Read and write, memory and storage | MStore, MLoad, SStore, SLoad |
| System | Message call into a contract | Call |

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
EIP-150 REVISION

https://ethereum.github.io/yellowpaper/paper.pdf

# System State

| | |
|---|---|
| Storage (S) | Persistent<br>Initially defined by the constructor |
| Memory (M) | Non-persistent<br>Reinitialized before every transaction |
| Stack (Q) | Limited to 1024 256-bit elements |
| Block Information (B) | Number, timestamp<br>Fixed for a given transaction |

# Contract Semantics

State: $\sigma = (S, M, Q, B)$

Transaction: $T = (\text{caller}, \{{}^{T}op_i\}, \ldots)$

Trace:

Final state

$$\sigma_0 \rightarrow \sigma_1 = {}^{T}op_0(\sigma_0) \rightarrow \ldots \rightarrow \sigma_{n-1} \rightarrow \sigma_n = {}^{T}op_n(\sigma_{n-1})$$

Stop

Semantics: set of all traces for this contract

# Unrestricted Writes

## Intuition

Anybody can execute owner = msg.sender

## Formalization

A write to o is unrestricted iff for any address a, there is

- T = (a, _)
- $\sigma_0 \rightarrow \sigma_1 =^{T} op_0(\sigma_0) \rightarrow \ldots \rightarrow \sigma_{i-1} \rightarrow \sigma_i =^{T} op_i(\sigma_{i-1}) \rightarrow \ldots$

with $op_i$ = SStore(o,_)

# Locked Ether

## Intuition

Payable function(s), but no transfer

## Formalization

There is a transaction increasing the balance:

- $\exists T.\ {}^{T}\sigma_0(\text{Balance}) < {}^{T}\sigma_n(\text{Balance})$

No transaction extracts ether:

- $\forall T.\ {}^{T}op_i = \text{Call}(\_,\_,x,\_) \Rightarrow x = 0$

# More Security Properties

Unexpected ether flows

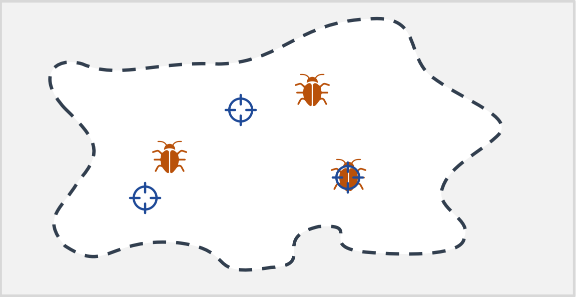
Insecure coding, such as unprivileged writes

Use of unsafe inputs (e.g., reflection, hashing, …)

Reentrant method calls (e.g., DAO bug)

Manipulating ether flows via transaction reordering

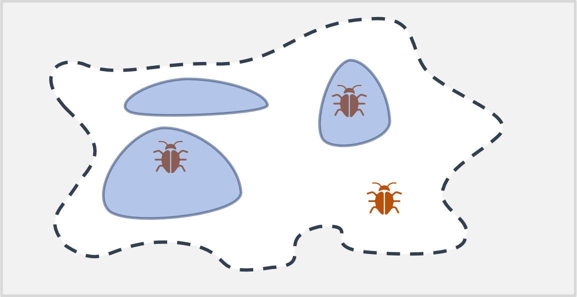
# Automated Techniques



Testing

Report true bugs

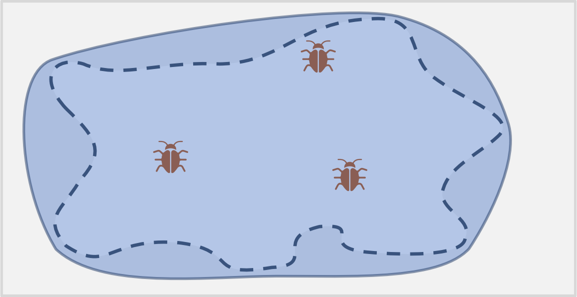
Can miss bugs



Dynamic Analysis

Report true bugs

Can miss bugs



Automated Verification

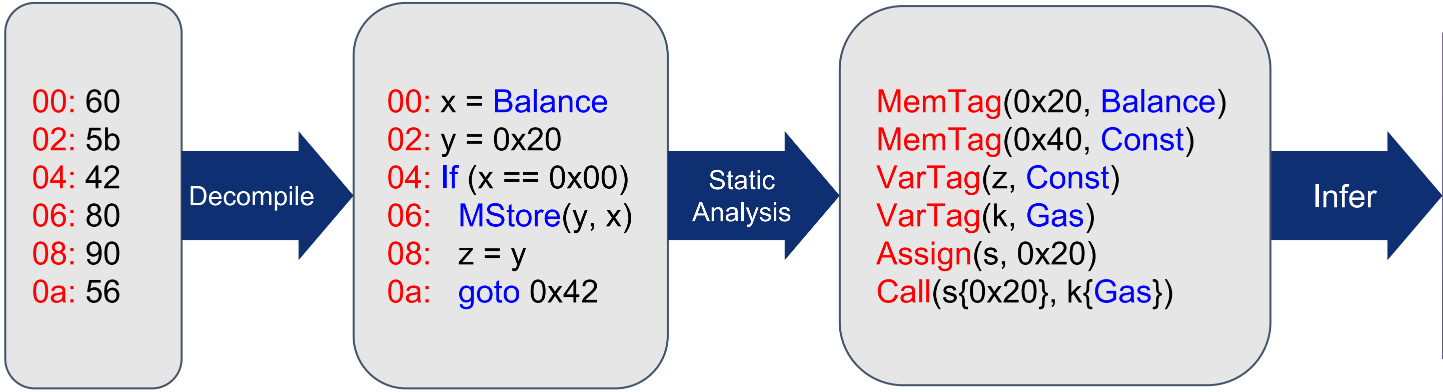
Can report false alarms

No missed bugs

Properties like unrestricted writes
cannot be checked on a single trace

# Demo

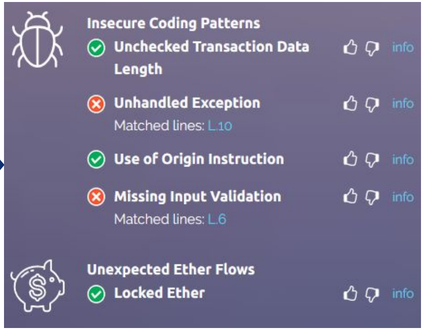
# Under the Hood



```
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
```

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



Insecure Coding Patterns

- ✓ Unchecked Transaction Data Length — info
- ⊗ Unhandled Exception — info
  Matched lines: L10
- ✓ Use of Origin Instruction — info
- ⊗ Missing Input Validation — info
  Matched lines: L6

Unexpected Ether Flows

- ✓ Locked Ether — info

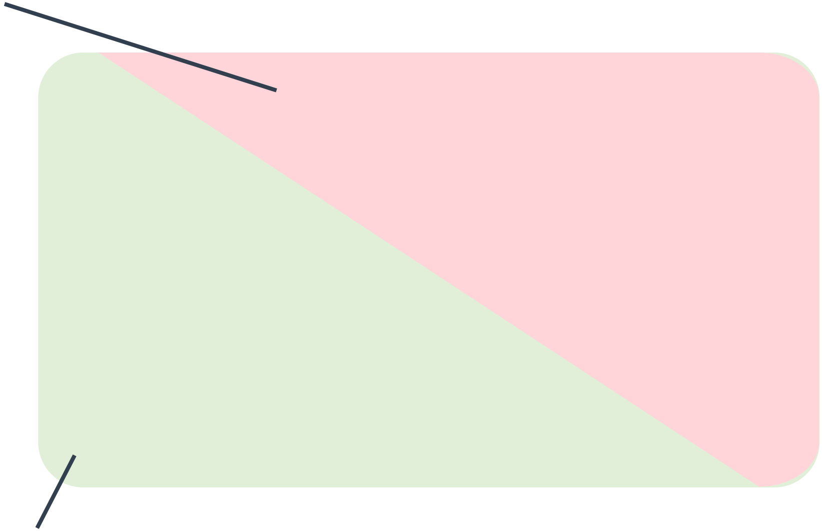
EVM
Binary

**Securify**
Intermediate
Representation

**Securify**
Semantic
Representation

**Securify** Report

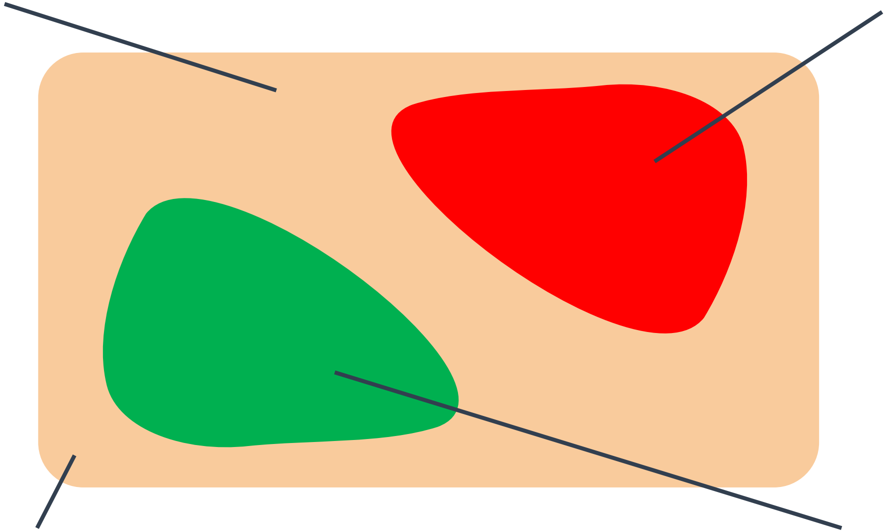
# Compliance and Violation Patterns



**Insecure** behaviors with respect to a property

**Secure** behaviors with respect to a property

# Compliance and Violation Patterns



**Violation** pattern

(under-approximates insecure behaviors)

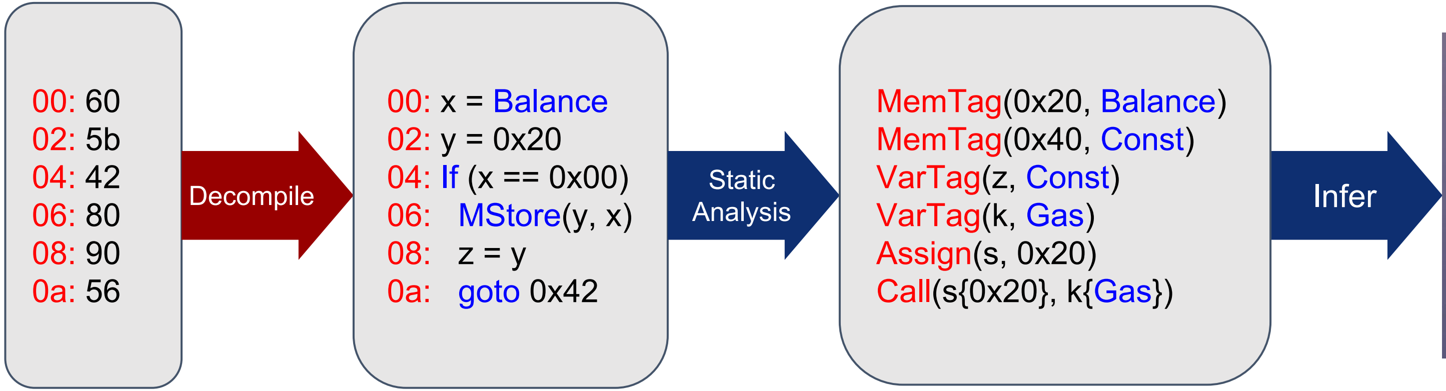
?

?

**Compliance** pattern

(under-approximates secure behaviors)

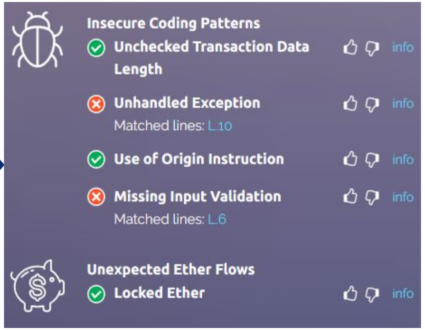
# Under the Hood: First Step



```
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
```

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



**Insecure Coding Patterns**

- ✓ **Unchecked Transaction Data Length** info
- ☒ **Unhandled Exception** info
  Matched lines: L.10
- ✓ **Use of Origin Instruction** info
- ☒ **Missing Input Validation** info
  Matched lines: L.6

**Unexpected Ether Flows**

- ✓ **Locked Ether** info

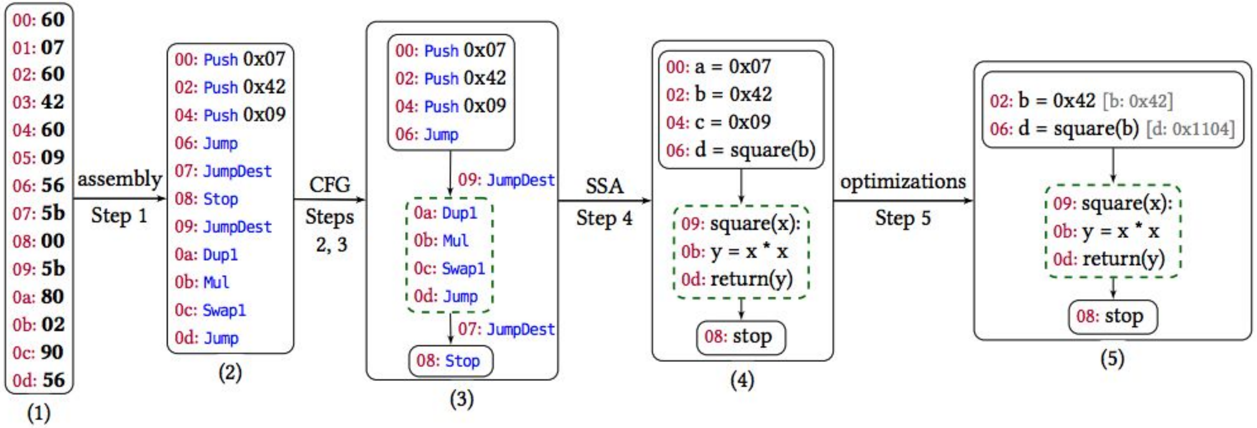
EVM
Binary

**Securify**
Intermediate
Representation

**Securify**
Semantic
Representation

**Securify** Report

# From EVM to CFG over SSA



```
00: 60
01: 07
02: 60
03: 42
04: 60
05: 09
06: 56
07: 5b
08: 00
09: 5b
0a: 80
0b: 02
0c: 90
0d: 56
```

(1)

assembly
Step 1

```
00: Push 0x07
02: Push 0x42
04: Push 0x09
06: Jump
07: JumpDest
08: Stop
09: JumpDest
0a: Dup1
0b: Mul
0c: Swap1
0d: Jump
```

(2)

CFG
Steps
2, 3

```
00: Push 0x07
02: Push 0x42
04: Push 0x09
06: Jump
```

09: JumpDest

```
0a: Dup1
0b: Mul
0c: Swap1
0d: Jump
```

07: JumpDest

```
08: Stop
```

(3)

SSA
Step 4

```
00: a = 0x07
02: b = 0x42
04: c = 0x09
06: d = square(b)
```

```
09: square(x):
0b: y = x * x
0d: return(y)
```

```
08: stop
```

(4)

optimizations
Step 5

```
02: b = 0x42 [b: 0x42]
06: d = square(b) [d: 0x1104]
```

```
09: square(x):
0b: y = x * x
0d: return(y)
```

```
08: stop
```

(5)

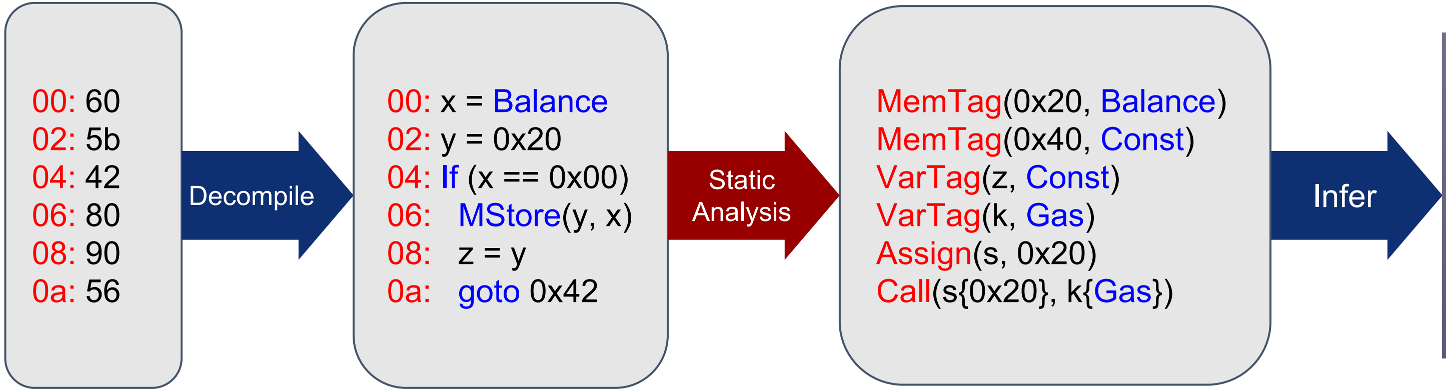
## Control flow graph (CFG)

- Node: a basic block
- Edge: jump from one basic block to another

## Static single assignment form (SSA)

- Each variable assigned exactly once

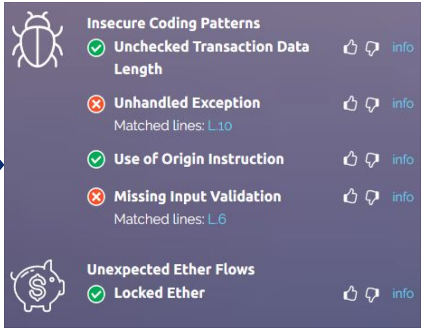
# Under the Hood: Second Step



```
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
```

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



**Insecure Coding Patterns**

- ✓ **Unchecked Transaction Data Length** info
- ⊗ **Unhandled Exception** info
  Matched lines: L10
- ✓ **Use of Origin Instruction** info
- ⊗ **Missing Input Validation** info
  Matched lines: L6

**Unexpected Ether Flows**

- ✓ **Locked Ether** info

EVM
Binary

**Securify**
Intermediate
Representation

**Securify**
Semantic
Representation

**Securify** Report

# Semantic Facts

| Semantic fact | Description |
|---|---|
| **Flow dependencies** | |
| MayFollow(pc, pc’) | The instruction at pc may follow that at pc’ |
| MustFollow(pc, pc’) | The instruction at pc must follow that at pc’ |
| **Data dependencies** | |
| MayDepOn(x, t) | The value of x may depend on tag t |
| MustDepOn(x, t) | The value of x must depend on tag t |
| DetBy(x, t) | For different values of t the value of x is different |

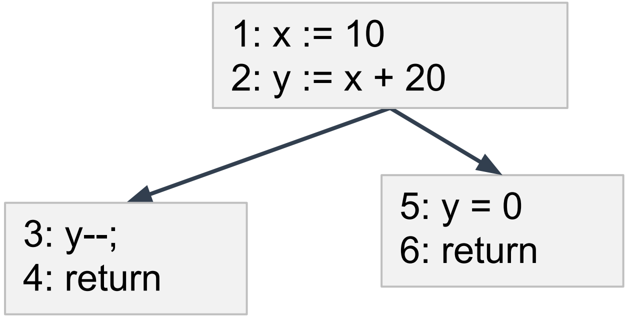
A tag can be an instruction or a variable

# Inference Rules: MayFollow

MayFollow(i, j) ← Follows(i, j)
MayFollow(i, j) ← Follows(i, k), MayFollow(k, j)

Derive input by declaring a predicate Follows(i, j) for:

- Edge (i, j) in the CFG
- Consecutive instructions in basic blocks



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

3: y--;
4: return

5: y = 0
6: return

Follows(1,2)
Follows(2,3)
Follows(3,4)
MayFollow(1,4)
Follows(2,5)
...

# Additional Input Facts

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

Code

```
Follows(1,2)
Follows(2,3)
Follows(3,4)

Assign(x, Balance)

IsConst(0x20)

MStore(2,0x20,x)

MLoad(3,y,0x20)

Op(4,z,x)
Op(4,z,y)
```

Input Facts

# Partial Inference Rules: MayDepOn

MayDepOn(x,t) ← Assign(x,t)

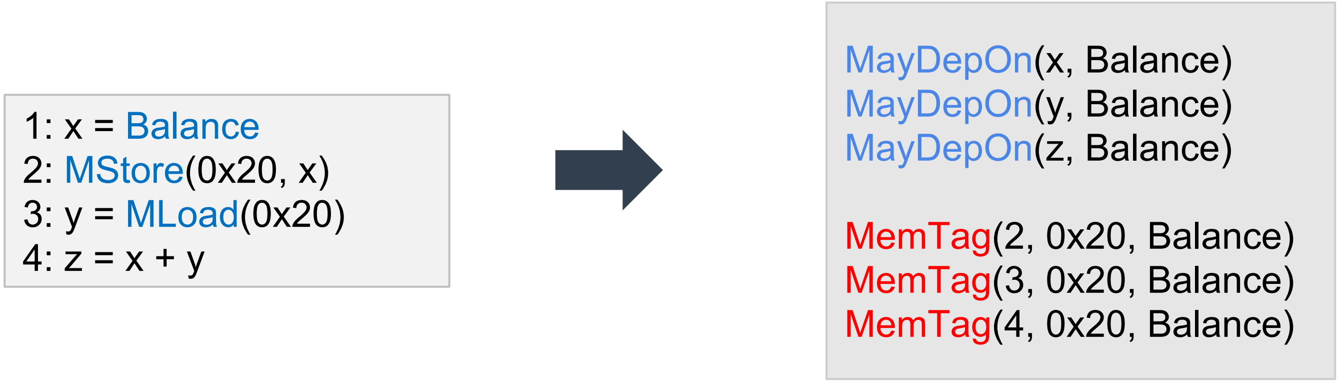
MayDepOn(x,t) ← Op(_,x,x'), MayDepOn(x',t)

MayDepOn(x,t) ← MLoad(l,x,o), isConst(o), MemTag(l,o,t)

MayDepOn(x,t) ← MLoad(l,x,o),¬isConst(o), MemTag(l,_,t)

- No label in MayDepOn
  - SSA form
- Label in MemTag
  - Offset dependencies evolve

# Derived Semantic Facts



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

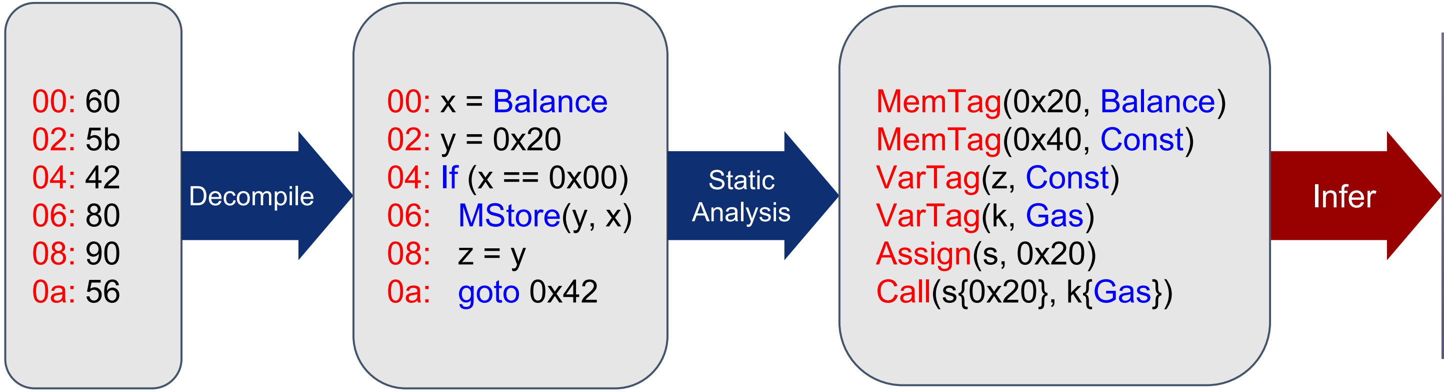
```
MayDepOn(x, Balance)
MayDepOn(y, Balance)
MayDepOn(z, Balance)

MemTag(2, 0x20, Balance)
MemTag(3, 0x20, Balance)
MemTag(4, 0x20, Balance)
```

Code

Derived semantic facts

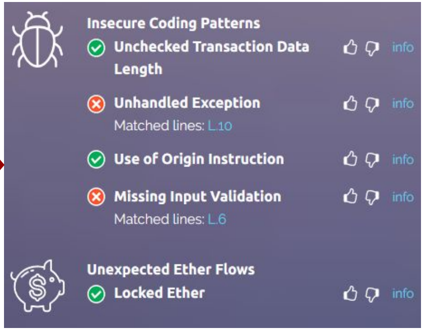
# Under the Hood: Final Step



```
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
```

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



Insecure Coding Patterns

✓ Unchecked Transaction Data Length — info

☒ Unhandled Exception — info
Matched lines: L10

✓ Use of Origin Instruction — info

☒ Missing Input Validation — info
Matched lines: L6

Unexpected Ether Flows

✓ Locked Ether — info

EVM Binary

**Securify** Intermediate Representation

**Securify** Semantic Representation

**Securify** Report

# Example Patterns: Restricted Write

Compliance pattern

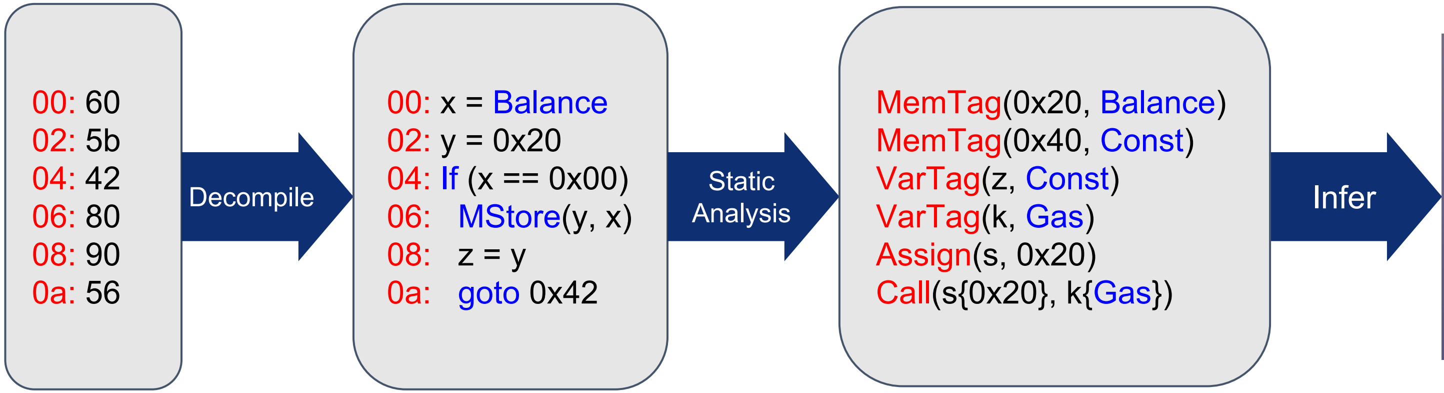
```
all SStore(l,o,_).DetBy(o, Caller)
```

Violation pattern

```
some SStore(l,o,_).
! MayDepOn(o, Caller) && ! MayDepOn(l, Caller)
```

- Remaining patterns are encoded similarly
- Proofs formally relate patterns and security properties

# Summary



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

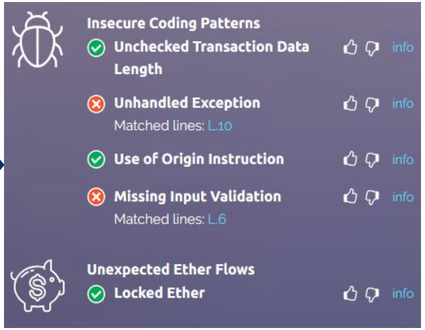
Decompile

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

Static Analysis

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

Infer



**Insecure Coding Patterns**

- ✓ Unchecked Transaction Data Length — info
- ☒ Unhandled Exception — info
  Matched lines: L.10
- ✓ Use of Origin Instruction — info
- ☒ Missing Input Validation — info
  Matched lines: L.6

**Unexpected Ether Flows**

- ✓ Locked Ether — info

EVM Binary

**Securify** Intermediate Representation

**Securify** Semantic Representation

**Securify** Report

## Research

SRL SOFTWARE RELIABILITY LAB

ICE center@ ETH

SECURIFY https://securify.ch

DEGUARD https://apk-deguard.com

JS NICE https://jsnice.org

PSI SOLVER https://psisolver.org

EVENT RACER https://eventracer.org

## Start-ups

CHAINSECURITY

https://chainsecurity.com

The first automated formal audit platform for smart contracts

WE'RE HIRING

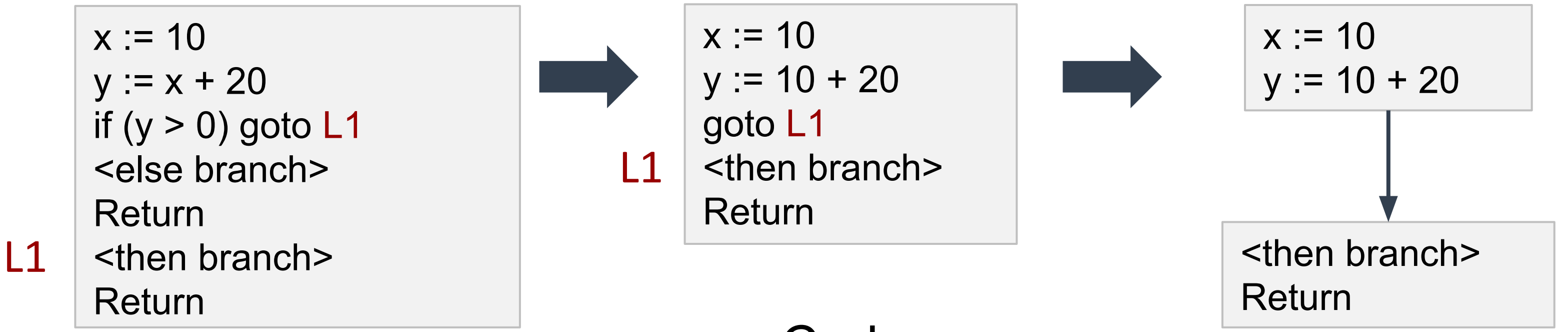
We are looking for strong business people and crypto experts to help our mission:

**jobs@chainsecurity.com**

✉ contact@chainsecurity.com

@chain_security

# Partial Evaluation



```
    x := 10
    y := x + 20
    if (y > 0) goto L1
    <else branch>
    Return
L1  <then branch>
    Return
```

Code

```
    x := 10
    y := 10 + 20
    goto L1
L1  <then branch>
    Return
```

Code
(partial evaluation)

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

```
<then branch>
Return
```

Constructed CFG

- Resolve jumps
  - Improve the precision of the CFG
- Resolve write offsets to storage / memory
  - Improve analysis precision

# Securify Pattern Language

| | | |
|---|---|---|
| Labels | l | (labels) |
| Vars | x | (variables) |
| Tags | t | l \| x |
| Instr | n | Instr(l,x,...,x) |
| Facts | f | MayFollow(l,l) \| MustFollow(l,l) \| MayDepOn(x,t) \| MustDepOn(x,t) \| DetBy(x,t) |
| Patterns | p | f \| all n.p \| some n.p \| p && p \| p \|\| p \| ! p |