Orc Secure Information Flow

Adrian Quark

January 30, 2009

Outline

- Classic solutions to secure information flow
- What makes Orc special
- Current approach

Big Picture

How to enforce policies on access to confidential data?
Applications:

- Automated business processes
  - handling credit cards
- Government agencies
  - handling social security numbers
- Security protocol implementations
  - handling private keys
- Software engineering
  - delegation

Channels

How can confidential data be leaked?

Direct channels:

- *Site calls*
  - Memory
  - File system
  - Inter-process communication
  - Network access

Covert channels:

- *Control flow*
- *Exceptions*
- *Termination*
- *Timing*
- Scheduler
- Cache behavior
- Resource exhaustion
- Power

Secure Information Flow

I'd tell you but then I'd have to kill you.

or

I'd tell you but then I'd have to lock you in my basement so you can't tell your friends.

or

While a process has knowledge of secure data, it must not pass it over insecure channels.

Secure Information Flow

Denning and Denning [1977].

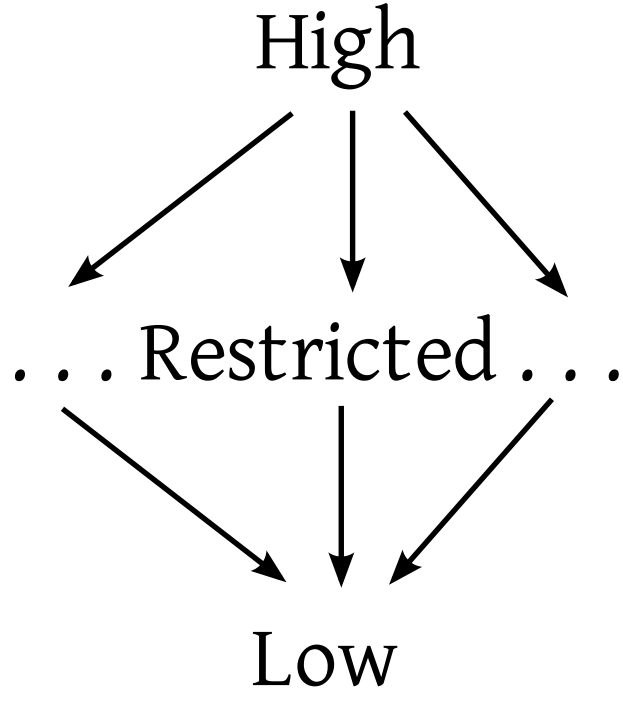
- **Security labels**: associated with input and output variables
- **Non-interference**: low-security inputs cannot affect high-security outputs
- **Information flow**: the value of one variable affects another

Security Labels

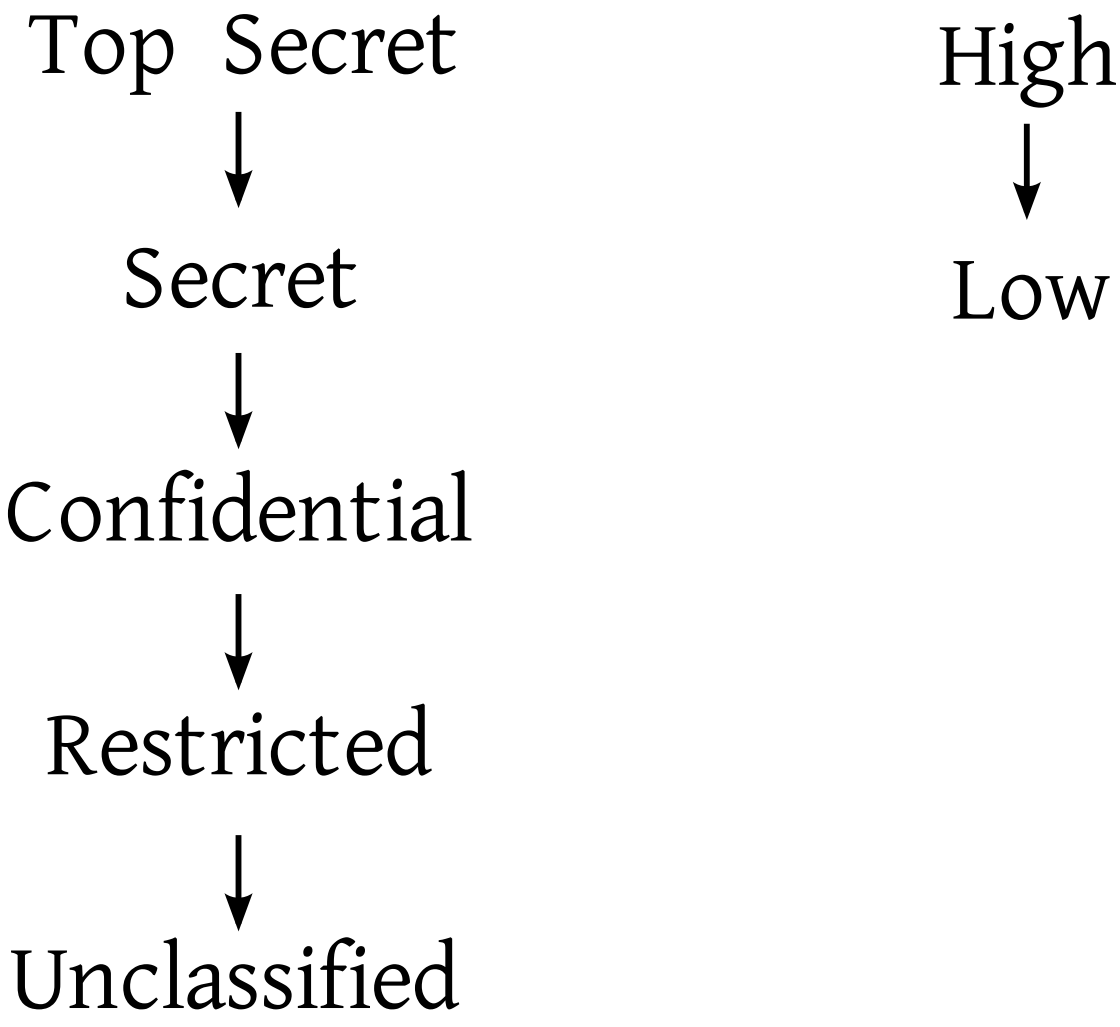
Labels form a lattice

Notation:

- $Low \sqsubseteq High$
- $Low \sqcup High = High$
- $v \in Low$



Example Security Labels



Non-interference

Program P denotes a function on states S:

- $P : S \to S \cup \{\bot\}$

Equivalence relation $=_L$:

- s_L is the projection in s of $v \in L' \sqsubseteq L$
- $s =_L s'$ iff $s_L = s'_L$

Non-interference: for all input states s and s'

- $s =_L s' \Rightarrow P(s) =_L P(s')$

More Non-interference

The semantic model may be generalized to allow richer observations

$$s =_L s' \Rightarrow P(s) \approx_L P(s')$$

where $\approx_L$ may consider

- *Set* of possible outputs (possibilistic)
- *Distribution* of possible outputs (probabilistic)
- Program traces (observational determinism or bisimulation)

Type System

Volpano, Smith and Irvine [1996] describe a type system for proving non-interference.

- Types are security labels
- Expressions are typed with the join of their constituents
- Statements are typed in a security context to prevent implicit information flow
- Well-typed terminating programs obey non-interference

Type System

Expression

$$\frac{\Gamma \vdash e : l \quad \Gamma \vdash e' : l'}{\Gamma \vdash e \star e' : l \sqcup l'}$$

Assignment

$$\frac{\Gamma(v) = l \quad \Gamma \vdash e : l' \quad \Gamma(\mathtt{pc}) \sqcup l' \sqsubseteq l}{\Gamma \vdash v := e}$$

Conditional

$$\frac{\Gamma \vdash e : l \quad \Gamma, \mathtt{pc} : l \vdash a \quad \Gamma, \mathtt{pc} : l \vdash b}{\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ a\ \mathtt{else}\ b}$$

Alternatives

- MAC
  - Requires full trust of those handling the data
- Dynamic SIF
  - Doesn't exploit language semantics
  - Requires trusted declassification kernels
  - May leak information through security failure

Orc

Good

- Parsimonious
- Functional
- Compositional

Bad

- Too expressive
- Sites are opaque
- Uses time and termination heavily

Examples

Memory

```
l := h
```

Examples

Control flow

```
if h then l := true else l := false
```

Examples

Dynamic security failure

```
l := false >>
if h then l := true else signal
```

Examples

Non-determinism

```
l := true | l := false | l := h
```

Examples

Compositionality

P:

```
h := true >>
Rtimer(10) >>
if h then l := true
     else l := false
```

Q:

```
Rtimer(5) >>
h := false
```

Examples

Internal Timing

```
( Rtimer(50) >> l := true
| (if h then Rtimer(100)
        else signal) >>
  l := false
)
```

Examples

External Timing

```
l := true >>
(if h then Rtimer(100)
      else signal) >>
l := false
```

Examples

Synchronization

```
Semaphore(0) >s>
l := false >>
( s.acquire() >> l := true
| if(h) >> s.release()
)
```

Examples

Non-termination

```
def loop(x) =
  if x then loop(x)
       else signal

( Rtimer(50) >> l := true
| loop(h) >> l := false
)
```

Current Approach

Generalize *pc* label used in static typing approach.

- Context: what can we infer at any program point?
- We can infer that all prior expressions published.
- Represent as a predicate in terms of program variables.
- Find the security level of the predicate.

Context

Values

```
val(~a, V)  = {not V(a)}
val(a && b, V) = {V(a) and V(b)}
val(a || b, V) = {V(a) or V(b)}
val(M(x), V) = {fresh symbol}
val(f >x> g, V) = {val(g, V+x:v) | v <- val(f, V)}
val(f <x< g, V) = {val(f, V+x:v) | v <- val(g, V)}
val(f | g, V) = val(f, V) U val(g, V)
val(f ; g, V) = val(f, V) U val(g, V)
```

Context

Publication

```
pub(if(x), P, V) = P(x) and (some V(x))
pub(M(x), P, V) = P(x) and (fresh symbol)
pub(f <x< g, P, V) = pub(f, P+x:pub(g, P),
                         V+x:value(g, V))
pub(f >x> g, P, V) = pub(f, P, V) and
                     pub(g, P+x:true,
                         V+x:value(f, V))
pub(f ; g, P, V) = pub(f, P, V) or pub(g, P, V)
pub(f | g, P, V) = pub(f, P, V) or pub(g, P, V)
```

Example 1

```
(if h then x := true
      else x := false) >>
l := true
```

Unsugared:

```
val t = h
( if(t) >> x := true
| if(~t) >> x := false
) >>
l := true
```

SECURE

Example 2

```
(if (l | h) then x := true
            else x := false) >>
l' := true
```

Unsugared:

```
val t = l | h
( if(t) >> x := true
  | if(~t) >> x := false
) >>
l' := true
```

SECURE

Example 3

```
(if h then if(l) >> x := true
      else x := false) >>
l' := true
```

Unsugared:

```
val t = h
( if(t) >> if(l) >> x := true
  | if(~t) >> x := false
) >>
{- (h and l) or (not h) -}
l' := true
```

INSECURE

More to Consider

- Compositionality [Bossi et al, 2007]
  - Low-level bisimulation requirement
  - Related to observational determinism
  - Can be used to eliminate timing channel
- Dynamic security labels [Zheng and Myers, 2004]
  - Type system looks like bounded polymorphism
  - Dynamic labels are necessary for most realistic applications