

Non-Interference for Interactive Programs

David Clark and Sebastian Hunt

Tuesday 21 August 2007

What is this talk about?

- What is the **right** setting for considering information flow security for interactive programs?
- We consider a recent paper that proposes *non-deducibility on strategies* in a language-based setting
- We investigate some variations and restrictions of the definitions
- The aim is to distinguish what is essential from what is not

Related work 1

- “Information flow in non-deterministic systems” Wittbold & Johnson [Security and Privacy 1990]
 - defined Non-Deducibility on Strategies: prohibits active channels as well as passive ones
 - synchronous statemachines setting
 - provided an example (originally due to Shannon)
 - satisfies non-deducibility on input strings **but not** non-deducibility on strategies

Related work 2

- “A logical approach to multilevel security of probabilistic systems” Gray & Syverson [Distributed Computing 1998]
- “Information flow security for interactive programs” O’Neill, Clarkson, & Chong [CSFW 2006]
 - “Non-Interference” properties for interactive programming languages
 - based on non-deducibility on *strategies*
 - setting is simple imperative language extended with non-deterministic and probabilistic choice

Framework

- A simple *while* language with input and output primitives
- A set of security types, \mathcal{L} ordered by \leq with $\tau \in \mathcal{L}$.
(we assume \mathcal{L} is the two point lattice $L \leq H$ for this talk)
- An *event* is the transmission of an input to or an output from the program:
- $\text{Ev}(\tau) \triangleq \{\text{in}(\tau, v), \text{out}(\tau, v) \mid v \in Z\}$
- $\text{Ev} \triangleq \bigcup_{\tau \in \mathcal{L}} \text{Ev}(\tau)$
- A *trace*, t , is a finite list of events

Operational semantics (selected rules)

$$[\text{Assign}] \frac{}{(x := e, \sigma, t, \omega) \longrightarrow (\text{skip}, \sigma[x := \sigma(e)], t, \omega)}$$

$$[\text{In}] \frac{\omega_\tau(t \upharpoonright \tau) = v}{(\text{input } x \text{ from } \tau, \sigma, t, \omega) \longrightarrow (\text{skip}, \sigma[x := v], t \hat{\wedge} \langle \text{in}(\tau, v) \rangle, \omega)}$$

$$[\text{Out}] \frac{\sigma(e) = v}{(\text{output } e \text{ to } \tau, \sigma, t, \omega) \longrightarrow (\text{skip}, \sigma, t \hat{\wedge} \langle \text{out}(\tau, v) \rangle, \omega)}$$

- ω is a pair of strategies ω_L, ω_H
- $t \upharpoonright \tau$ denotes t with all non- τ events removed

Shannon Example [OCC]

```
while (true) do
  x := 0 | x := 1;
  output x to H;
  input y from H;
  output (x xor y) to L;
```

High's Strategy 101

$i_H(y)$ abbreviates $\text{in}(H, y)$, $o_H(x)$ abbreviates $\text{out}(H, x)$, etc

$$\begin{aligned}\omega(\langle o_H(x) \rangle) &= x \text{ xor } 1 \\ \omega(\langle o_H(x) i_H(y) o_L(z) o_H(x') \rangle) &= x' \text{ xor } 0 \\ \omega(\langle o_H(x) i_H(y) o_L(z) o_H(x') i_H(y') o_L(z') o_H(x'') \rangle) &= x'' \text{ xor } 1\end{aligned}$$

Transmits 101 to Low, even though Low has no idea what High actually inputs.

Non-deducibility on strategies

- Security condition is an instance of *non-deducibility on strategies* as defined by Wittbold & Johnson
- A program is secure if, for every initial state, σ , any trace of events seen on channel L is consistent with every possible user strategy for channel H .
- Shannon example clearly does *not* have this property

Non-deducibility on strategies

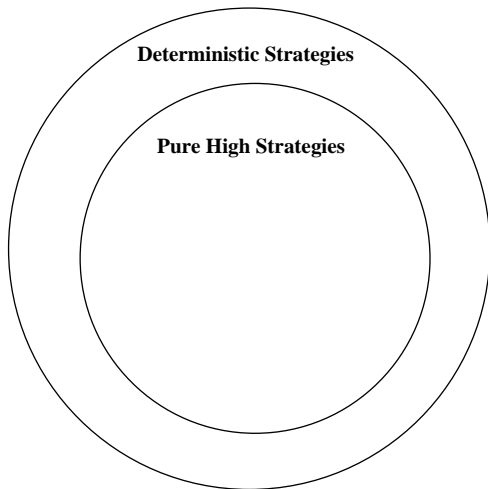
- For $m = (c, \sigma, \langle \rangle, \omega)$ we write $m \rightsquigarrow t$ to mean $\exists c', \sigma'. (c, \sigma, \langle \rangle, \omega) \rightarrow^* (c', \sigma', t, \omega)$
- we write $t =_{\tau} t'$ to mean $t \upharpoonright \tau = t' \upharpoonright \tau$
- [OCC 2006] A command c satisfies non-deducibility on strategies iff for all $m = (c, \sigma, \langle \rangle, \omega)$ and $m' = (c, \sigma, \langle \rangle, \omega')$:

if $\omega_L = \omega'_L$ and $m \rightsquigarrow t$
then $\exists t' . t =_L t'$ and $m' \rightsquigarrow t'$

What is a strategy?

- A mapping from traces to values (inputs)
- OCC define ω_H as a function on traces of H -events and ω_L as a function on traces of L -events
- What happens if:
 - we allow High to see L -events?
 - we allow strategies to be non-deterministic? probabilistic?
 - we restrict strategies even further (streams)?

Pure High Strategies



Unrestricted deterministic strategies

- Slight reformulation of the semantics
- Strategies get applied to the full trace so far:

$$[\text{In}] \frac{\omega_{\tau}(t) = v}{(\text{input } x \text{ from } \tau, \sigma, t, \omega) \longrightarrow (\text{skip}, \sigma[x := v], t \hat{\ } \langle \text{in}(\tau, v) \rangle, \omega)}$$

Low strategies and Pure High strategies

- Low strategies can see only L -events:

$$t =_L t' \Rightarrow \omega_L(t) = \omega_L(t')$$

- A High strategy is *pure* if it can see only H -events:

$$t =_H t' \Rightarrow \omega_H(t) = \omega_H(t')$$

Streams as strategies

- Define $t =_{\tau}^S t'$ iff
t and t' contain the same number of τ -input events
- A strategy ω is a τ -stream iff:

$$t =_{\tau}^S t' \Rightarrow \omega(t) = \omega(t')$$

Streams Suffice for Deterministic Programs

Proposition:

For deterministic programs it makes no difference whether we define Non-Deducibility with respect to arbitrary non-deterministic High strategies, arbitrary deterministic High strategies, Pure High strategies or Streams.

Proof Sketch (Part 1)

A command c fails non-deducibility on strategies iff there exist some $\sigma, \omega_L, \omega_H, \omega'_H$ such that:

$$(c, \sigma, \langle \rangle, (\omega_H, \omega_L)) \rightsquigarrow t$$

but $\forall t' . (c, \sigma, \langle \rangle, (\omega'_H, \omega_L)) \rightsquigarrow t' \text{ implies } t \neq_L t'$

- Given such a counterexample with deterministic strategies, construct stream-strategies $\widehat{\omega}_L, \widehat{\omega}_H, \widehat{\omega}'_H$ such that:
 - $(c, \sigma, \langle \rangle, (\widehat{\omega}_H, \widehat{\omega}_L)) \rightsquigarrow t$
 - if $(c, \sigma, \langle \rangle, (\widehat{\omega}'_H, \widehat{\omega}_L)) \rightsquigarrow t'$ then $t \neq_L t'$
- Non-deterministic case dealt with separately (easy)

Proof Sketch (Part 2)

$$\begin{aligned}\widehat{\omega}_H(t) &= \begin{cases} v & \text{if } \exists t' =_H^S t.(c, \sigma, \langle \rangle, (\omega_H, \omega_L)) \rightsquigarrow t' \wedge \langle in(H, v) \rangle \\ 0 & \text{otherwise} \end{cases} \\ \widehat{\omega}'_H(t) &= \begin{cases} v & \text{if } \exists t' =_H^S t.(c, \sigma, \langle \rangle, (\omega'_H, \omega_L)) \rightsquigarrow t' \wedge \langle in(H, v) \rangle \\ 0 & \text{otherwise} \end{cases} \\ \widehat{\omega}_L(t) &= \begin{cases} v & \text{if } \exists t' =_L^S t.(c, \sigma, \langle \rangle, (\omega_H, \omega_L)) \rightsquigarrow t' \wedge \langle in(L, v) \rangle \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

We can show:

- well-defined (for deterministic c)
- behave as required for the proof

Future Work

- Deterministic programs + probabilistic strategies?
- Non-deterministic programs:
 - Shannon example shows that streams do *not* suffice
 - OCC consider Pure High strategies
 - What happens if we consider arbitrary High strategies?
probabilistic High strategies?
 - Can we always transform out the non-determinism by adding additional parameters?