Observational Determinism for Concurrent Program Security

Steve Zdancewic  Andrew Myers
Computer Security Foundations Workshop 2003

Security conditions vs. analyses

- The security-typing game:
  1. An intuitive semantic security condition that guarantees behavior of program is secure
  2. A static program analysis (type system) that ensures the program obeys the security condition
- Useful if:
  - Security condition corresponds to desired security
  - Analysis permits interesting programs

Noninterference

- Definitions:
  \( (M, e) \) : a configuration (memory \( M \), program \( e \))
  \( (M, e) \Uparrow T \) : configuration \( (M, e) \) executes with result \( T \)
  \( T_1 \approx_1 T_2 \) : 'low observer' at \( L \) can't distinguish results
  \( (M_1, e_1) \approx_L (M_2, e_2) \) : can't distinguish inputs
- Noninterference:
  \( (M_1, e_1) \Uparrow T_1 \Rightarrow T_1 \approx_L T_2 \)

Possibilistic problems

- \( l := \text{false} | l := \text{true} | l := h \)
- Random scheduling: 2/3 probability leak
- Sequential scheduling: ~1 probability leak
- High information communicated via scheduler
- Possibilistically "secure"
  \( h = \text{false} \Rightarrow (1 = \text{false}, l = \text{true}) \)
  \( h = \text{true} \Rightarrow (1 = \text{false}, l = \text{true}) \)
- Information "leaked" only if attacker is certain
- Nondeterminism doesn’t work against the attacker!
Timing channels

- Time taken by program can reveal sensitive information
- Can be converted into storage channels
- Random scheduling; possibilistically "secure"
- One solution: consider time observable
- Problem: rejects secure sequential programs

```
x := true;
sleep(50);
l := x  
if h then sleep(100)  
else skip;
x := false
```

Effect: \( l := h \)

Problems to solve

1. Possibilistic security: insecure
   - Need a stronger security condition that’s not...

2. Ruling out all timing channels: too restrictive
   - Need a weaker security condition (& analysis)

1. A "new" security condition

- Observational determinism [McLe92, Rosc95]:
  \( \langle M_i, e_i \rangle \Downarrow T_i \Rightarrow T_1 \approx_L T_2 \)
- Any observable difference between outputs permits a refinement attack
- System may still be nondeterministic - depends on choice of \( T_i \approx_L \)

Controlling internal channels

- Insight: Internal timing channels require races
- Write-write race:
  \( l := \text{false} \mid l := \text{true} \mid l := h \)
- Read-write race:
  \( \text{sleep}(50); l := x \mid \text{if } h \text{ then } \text{sleep}(100) \mid \text{else skip;} \mid x := \text{false} \)
- Race = two memory accesses to same location, at least one a write, that can occur in either order
- Observational determinism \( \Rightarrow \) rule out races
- Nondeterminism ok at different locations

2. Avoiding restrictiveness

- Idea: distinguish between internal and external timing channels
  - Internal: affect program data
  - External: affect only timing of external interactions

```
x := true;
sleep(50);
l := x  
if h then sleep(100)  
else skip;
x := false
```

Limiting observational power

- Idea: capture invisibility of external timing channels in relation \( T_1 \approx_L T_2 \)
- Result of concurrent computation is trace \( T \) of memory states \([M_1, M_2, M_3, \ldots]\)
- Projection of \( T \) onto location \( l \) is
  \( T(l) = [M_1(l), M_2(l), \ldots] \)
- Traces are indistinguishable if they look the same at every memory location
  - Can’t time updates
  - Can’t time execution

```
x := true;
sleep(50);
l := x  
if h then sleep(100)  
else skip;
x := false
```

\( T_1(l) = [v_1, v_2, v_3, v_4, \ldots] \approx_L T_2(l) = [v_1, v_2', v_3', v_4', \ldots] \)
Synchronization

- Races considered harmful!
- Unsynchronized writes to shared memory unsafe
  ⇒ need synchronization and communication mechanisms
- Our choice: message passing (blocking snd/rcv)
  \[ \text{snd}(c,v) \rightarrow x := \text{rcv}(c); \]
- Supports non-block-structured communication
- Shared memory, but restricted to prevent unsynchronized communication

A secure concurrent language

- Variant of the Join calculus [Fournet et al.]
  - Explicit message passing
  - High-level abstraction for synchronization
  - Similar to Milner’s π-calculus
- Explicit state using ML-style references
  Use an alias analysis that prevents races
- Linear/Nonlinear channels
  - Adapted from linear continuations [ZM’02]
  - Regulates communication & synchronization between threads
  - See also [Honda & Yoshida ’02]

Details

\[
J ::= f(x,y) \quad \text{nonlinear channels} \\
f(x) \quad \text{linear channels} \\
(J|J) \quad \text{join patterns}
\]

\[
P ::= \text{let } x = \text{ref } v \text{ in } P \quad \text{ref creation} \\
| \text{set } v := v \text{ in } P \quad \text{ref assignment} \\
| \text{let } J \triangleright P \text{ in } P \quad \text{chan. defn.} \\
| \text{let } J \rightarrow P \text{ in } P \quad \text{lin. chan. defn.} \\
| \text{if } v \text{ then } P \text{ else } P \quad \text{conditional} \\
| v(v,l) \quad \text{msg. send} \\
| l(v) \quad \text{lin. msg. send} \\
| (P|P) \quad \text{parallel comp.} \\
| 0 \quad \text{inactive proc.}
\]

Example

\[
\text{let } \text{double}(c) \triangleright c() \mid c() \text{ in} \\
\text{let } d() \triangleright P \text{ in} \\
\text{P()} \mid d()
\]

Linear channels

Concurrent threads may not have races.

Linear synchronization and race-freedom imply that that \( P \) learns no information about how the scheduler executed \( R_0 \) and \( R_1 \).
Linear message passing

The type system also permits rich kinds of synchronization behavior.
Previous type systems reject processes with this synchronization structure.
[Honda et al. ’02] [Pottier ’02] [Sabelfeld ’01]

External channels

- Memory locations are externally observable
- Can encode external I/O channels
- Limited observational power
  \[ \Rightarrow \text{external I/O channels can’t be timed against each other} \]

Shared memory vs message-passing

- Shared-memory programming model:
  - Common shared memory locations used for mutation, communication
  - Synchronization: locks/semaphores, condition variables
- Locks don’t help!
  \[ l := \text{false} \mid l := \text{true} \mid l := h \]
- Shared-memory model is fundamentally uncongenial to information flow analysis

Compositionality

- Connecting secure programs with communication channels isn’t secure in general
- Composition is in the language
  - Channels must agree on security labels
  - Composition must not introduce races

Future work

- Need a good race freedom analysis
  - Ideally, compositional (but what annotations?)
- Application to practical language (Jif?)
- Handle lock/semaphore synchronization