Language-Based Security

Lecture 3: Information Flow Enforcement

Stephen Chong, Harvard University
Road Map

• Intro
  • Formal Methods for Security
  • Language-Based Security
  • Case Study: Noninterference

• Primer on Computer Security

• Information Flow
  • Semantics
    • Enforcement
    • Beyond confidentiality

• Enforcing Language Abstractions
Enforcement of Information Flow
From Semantics To Enforcement

- We have discussed semantics of information flow
- Very carefully separated from enforcement mechanism
  - I.e., defining our notion of security without how we are going to enforce it
- Let’s consider how to enforce noninterference, i.e., control the flow of information in systems
Dimensions of Enforcement

- Enforcement mechanisms differ on granularity and when enforcement occurs.

Granularity:
- **Coarse grained** mechanisms track information at granularity of computational containers:
  - Contains both code and data
  - Different granularity of containers, e.g., process, function, block scope, ...
- **Fine grained** mechanisms track information at level of values/variables

When does enforcement happen?
- **Static** mechanisms enforce security before execution
- **Dynamic** mechanisms enforce security during execution
  - (Hybrid mechanisms use a combination)

In this lecture, we will look briefly at:
- Security type system (static fine-grained)
- Fine-grained information-security monitor (dynamic fine-grained)
- Coarse-grained information-security monitor (dynamic coarse-grained)
Security-Typed Language

- Type system to enforce (fine-grained) information flow
- Let’s see the key ideas in IMP
- Two judgments:

\[ \Gamma \vdash e : \tau_\ell \]
\[ \Gamma, pc \vdash c \]

Context \( \Gamma \) maps vars to labeled types, \( \tau_\ell \)

Expression (boolean or arithmetic)

Labeled type
\[ \tau ::= \text{int} \mid \text{bool} \]
\( \ell \in \Lambda \)
Label is upper bound on info that influences the value
Typing of Expressions

\[ \Gamma \vdash e : \tau_\ell \]

\[
\begin{align*}
\Gamma \vdash n &: \mathbb{int}_\perp \\
\Gamma \vdash true &: \mathbb{bool}_\perp \\
\Gamma \vdash false &: \mathbb{bool}_\perp \\
\Gamma \vdash x &: \Gamma(x)
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash a_1 &: \mathbb{int}_{\ell_1} &\quad \Gamma \vdash a_2 &: \mathbb{int}_{\ell_2} &\quad \ell = \ell_1 \sqcup \ell_2 \\
\Gamma \vdash a_1 + a_2 &: \mathbb{int}_\ell \\
\Gamma \vdash a_1 < a_2 &: \mathbb{bool}_\ell &\quad \ell = \ell_1 \sqcup \ell_2
\end{align*}
\]
Typing of Commands

$\Gamma, pc \vdash c$

- Context $\Gamma$ maps vars to labeled types, $\tau_\ell$
- Command
- $pc \in \Lambda$
  - $Program\ counter\ level$
  - (1) a lower bound on the side effects of $c$
  - (2) an upper bound on the info that affects whether this command is executed
Typing of Commands

\[
\Gamma, pc \vdash c
\]

\[
\begin{array}{c}
\Gamma, pc \vdash \text{skip} \\
\Gamma \vdash e : \tau_{\ell_e} \quad \ell_e \sqsubseteq \ell_x \\
\Gamma, pc \vdash x := e \\
\Gamma, pc \vdash c_1 \\
\Gamma, pc \vdash c_2
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash b : \text{bool}_{\ell} \\
\Gamma, pc \sqcup \ell \vdash c_1 \\
\text{if } b \text{ then } c_1 \text{ else } c_2
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash b : \text{bool}_{\ell} \\
\Gamma, pc \sqcup \ell \vdash c
\end{array}
\]

\[
\begin{array}{c}
\Gamma, pc \vdash \text{while } b \text{ do } c
\end{array}
\]
Examples

\[ \text{sec} := \text{pub} + 42; \]

\[
\frac{
\frac{
\Gamma \vdash \text{pub} : \text{int}_L \quad \Gamma \vdash 42 : \text{int}_L
}{
\Gamma \vdash \text{pub} + 42 : \text{int}_L
}\quad \frac{
\Gamma \vdash \text{pub} + 42 : \text{int}_L
}{
\Gamma, L \vdash \text{sec} := \text{pub} + 42
\}
}{
\frac{
\Gamma \vdash e : \tau_{\ell_e} \quad \ell_e \sqcup pc \sqsubseteq \ell_x
}{
\Gamma, pc \vdash x := e
}\quad \frac{
\Gamma(x) = \tau_{\ell_x}
}{
\frac{
\frac{
\frac{
\Gamma \vdash \text{pub} : \text{int}_L \quad \Gamma \vdash 42 : \text{int}_L
}{
\Gamma \vdash \text{pub} + 42 : \text{int}_L
}\quad \frac{
\Gamma \vdash \text{sec} : \text{int}_H \quad \Gamma \vdash 42 : \text{int}_L
}{
\Gamma \vdash \text{sec} + 42 : \text{int}_H
\}
}{
\frac{
\Gamma \vdash \text{sec} + 42 : \text{int}_H
}{
\Gamma \vdash \text{pub} := \text{sec} + 42
\}
}\quad \frac{
\Gamma \vdash \text{pub} := \text{sec} + 42
}{
\Gamma, L \vdash \text{pub} := \text{sec} + 42
\}
}\quad \frac{
\Gamma \vdash \text{pub} := \text{sec} + 42
}{
\Gamma, L \vdash \text{pub} := \text{sec} + 42
\}
}
\]
Examples

if (sec < 0)
sec := −sec

Γ ⊢ \text{sec < 0} : \text{bool}_H
Γ ⊢ \text{sec < 0} : \text{int}_H
Γ, H ⊢ \text{sec := −sec}
Γ, H ⊢ \text{skip}
Γ, L ⊢ \text{if sec < 0 then sec := −sec else skip}

if (sec < 0)
pub := 42

Γ ⊢ \text{pub := 42}
Γ ⊢ \text{42 : int}_L
Γ, H ⊢ \text{pub := 42}
Γ, H ⊢ \text{skip}
Γ, L ⊢ \text{if sec < 0 then pub := 42 else skip}
Soundness of Type System

• Theorem: For all programs c, if $\Gamma, \perp \vdash c$ then c is noninterfering, i.e.,

$\text{For all } \sigma_1, \sigma_2, \sigma_1', \sigma_2', \ell$

if $\sigma_1 =_\ell \sigma_2$ and $\langle c, \sigma_1 \rangle \Downarrow \sigma_1'$ and $\langle c, \sigma_2 \rangle \Downarrow \sigma_2'$

then $\sigma_1' =_\ell \sigma_2'$

• Proof:

• Lots of techniques possible for proving relational properties
• Direct proof based on induction (on large step operational semantics)
• Logical relations
• “Squared” language approach (Due to Pottier & Simonet, 2003)
  • Create a language IMP$^2$ where one execution of an IMP$^2$ represents 2 IMP executions
  • ...

Stephan Chong, Harvard University
Another Type System

\[ e ::= x \mid n \mid () \mid e_1 \ e_2 \mid \lambda x : \tau, \ell. \ e \mid \text{input from } \ell \mid \text{output } e \text{ to } \ell \mid \text{let } x = e_1 \ \text{in } e_2 \]

\[ \sigma ::= \text{unit} \mid \text{int} \mid \tau_1 \xrightarrow{pc} \tau_2 \]

\[ \tau ::= \sigma \ell \]

**Latent effect** program counter label

- Is lower bound of side effects of function body
- Is the pc label used to type check function body

Labeled type.

(Label is upper bound on info that influences value of base type \( \sigma \))
Another Type System

\[ e ::= x \mid n \mid () \mid e_1 \ e_2 \mid \lambda x : \tau, \ell. \ e \]
\[ \mid \text{input from } \ell \mid \text{output } e \text{ to } \ell \]
\[ \mid \text{let } x = e_1 \text{ in } e_2 \]

\[ \sigma ::= \text{unit} \mid \text{int} \mid \tau_1 \xrightarrow{pc} \tau_2 \]
\[ \tau ::= \sigma_\ell \]

\[ \Gamma, pc \vdash x : \Gamma(x) \sqcup pc \]
\[ \Gamma, pc \vdash n : \text{int}_{pc} \]
\[ \Gamma, pc \vdash () : \text{unit}_{pc} \]

\[ \sigma_\ell \sqcup \ell' \triangleq \sigma_{\ell \sqcup \ell'} \]
Another Type System

e ::= x | n | () | e_1 e_2 | \lambda x: \tau, \ell.e
| input from \ell | output e to \ell
| let x = e_1 in e_2

\sigma ::= \text{unit} | \text{int} | \tau_1 \xrightarrow{pc} \tau_2
\tau ::= \sigma_{\ell}

\Gamma, pc \vdash x : \Gamma(x) \sqcup pc
\Gamma, pc \vdash n : \text{int}_{pc}
\Gamma, pc \vdash () : \text{unit}_{pc}

\Gamma, pc \vdash \lambda x : \tau, \ell.e : (\tau \xrightarrow{\ell} \tau')_{pc}
\Gamma, pc \vdash e_1 : (\tau \xrightarrow{pc_1} \tau')_{\ell_1}
\Gamma, pc \vdash e_2 : \tau
\ell_1 \sqcup pc \sqsubseteq pc_1

\Gamma, pc \vdash \text{input from} \ell : \text{int}_{\ell \sqcup pc}

\text{Input is a side effect at level } \ell, \text{ so } pc \text{ must be a lower bound}

\text{Info leading to decision to execute the function body}
Another Type System

\[
e ::= x \mid n \mid () \mid e_1 \ e_2 \mid \lambda x: \tau, \ell. \ e
\]

| \text{input from } \ell | \text{output } e \text{ to } \ell | \text{let } x = e_1 \text{ in } e_2
| \sigma ::= \text{unit} \mid \text{int} \mid \tau_1 \xrightarrow{pc} \tau_2 |
| \tau ::= \sigma_\ell |

\[
\Gamma, pc \vdash x: \Gamma(x) \sqcup pc
\]

\[
\Gamma, pc \vdash n: \text{int}_{pc}
\]

\[
\Gamma, pc \vdash (): \text{unit}_{pc}
\]

\[
\Gamma, pc \vdash \lambda x: \tau, \ell. \ e: (\tau \xrightarrow{\ell} \tau')_{pc}
\]

\[
\Gamma, pc \vdash e_1: (\tau \xrightarrow{pc_1} \tau')_{\ell_1} \quad \Gamma, pc \vdash e_2: \tau \quad \ell_1 \sqcup pc \sqsubseteq pc_1
\]

\[
\Gamma, pc \vdash e_1 \ e_2: \tau' \sqcup pc
\]

\[
\Gamma, pc \vdash \text{input from } \ell: \text{int}_{\ell \sqcup pc}
\]

\[
\Gamma, pc \vdash e: \tau \quad \tau \leq \tau'
\]

\[
\sigma \leq \sigma' \quad \ell \sqsubseteq \ell'
\]

\[
\sigma_\ell \leq \sigma'_{\ell'}
\]
Another Type System

\[
e ::= x \mid n \mid () \mid e_1 \, e_2 \mid \lambda x : \tau, \ell. e
\]

\[
\text{input from } \ell \text{ | output } e \text{ to } \ell
\]

\[
\text{let } x = e_1 \text{ in } e_2
\]

\[
\sigma ::= \text{unit} \mid \text{int} \mid \tau_1 \xrightarrow{pc} \tau_2
\]

\[
\tau ::= \sigma_{\ell}
\]

\[
\Gamma, pc \vdash x : \Gamma(x) \sqcup pc
\]

\[
\Gamma, pc \vdash \lambda x : \tau, \ell. e : (\tau \xrightarrow{\ell} \tau')_{pc}
\]

\[
\Gamma, pc \vdash \text{input from } \ell : \text{int}_{\ell \sqcup pc}
\]

\[
\Gamma, pc \vdash \text{output } e \text{ to } \ell : \text{unit}_{\ell \sqcup pc}
\]

\[
\Gamma, pc \vdash n : \text{int}_{pc}
\]

\[
\Gamma, pc \vdash () : \text{unit}_{pc}
\]

\[
\Gamma[x \mapsto \tau], \ell \vdash e : \tau'
\]

\[
\Gamma, pc \vdash e_1 : (\tau \xrightarrow{pc_1} \tau')_{\ell_1}
\]

\[
\Gamma, pc \vdash e_2 : \tau
\]

\[
\ell_1 \sqcup pc \sqsubseteq pc_1
\]

\[
\Gamma, pc \vdash e_1 \, e_2 : \tau' \sqcup pc
\]

\[
\sigma \leq \sigma'
\]

\[
\ell \sqsubseteq \ell'
\]

\[
\sigma_{\ell} \leq \sigma'_{\ell'}
\]

\[
\tau_1' \leq \tau_1
\]

\[
\tau_2 \leq \tau_2'
\]

\[
pc' \sqsubseteq pc
\]

\[
\tau_1 \xrightarrow{pc} \tau_2 \leq \tau_1' \xrightarrow{pc'} \tau_2'
\]

\[
\ldots
\]
Other Language Features

• Can extend basic ideas of security type system for other language features
  • References (i.e., first-class memory)
  • Exceptions
    • Track information flow associated with normal termination or exceptional termination
  • First-class Labels
  • ...

Stephen Chong, Harvard University
Fine-Grained Dynamic Enforcement

• Dynamic enforcement techniques monitor and restrict execution at runtime
  • Mechanism modifies program behavior! It is an information channel!
  • Need to be aware of what information it reveals by (not) intervening
  • May need to adapt the security condition to account for additional observations
Dynamic Info Flow Tracking

- Flow-Insensitive:

\[
\begin{align*}
pc &\sqcup \Gamma(pub + 42) \sqsubseteq \Gamma(sec) \\
L &\sqcup L \sqsubseteq H
\end{align*}
\]

\[
\begin{align*}
sec &\Rightarrow L \\
sec &\Rightarrow H \\
pub &\Rightarrow L
\end{align*}
\]

\[
\begin{align*}
sec &:= pub + 42; \\
pub &:= pub + 7; \\
pub &:= sec; \\
pub &:= 42
\end{align*}
\]
Dynamic Info Flow Tracking

- Flow-Insensitive:

```
sec := pub + 42;
pub := pub + 7;
pub := sec;
pub := 42
```

\[
egin{align*}
pc &\sqsubseteq (\Gamma(pub+7) \sqsubseteq \Gamma(pub)) \\
L &\sqsubseteq (L \sqsubseteq L)
\end{align*}
\]

\[
egin{align*}
\text{sec} &\Rightarrow H \\
\text{pub} &\Rightarrow L
\end{align*}
\]
Dynamic Info Flow Tracking

• Flow-Insensitive:

\[ pc \sqcup \Gamma(\text{sec}) \subseteq \Gamma(\text{pub}) \]
\[ L \sqcup H \subseteq L \]

\[ pc \mapsto L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]

\[ \text{sec} := \text{pub} + 42; \]
\[ \text{pub} := \text{pub} + 7; \]
\[ \text{pub} := \text{sec}; \]
\[ \text{pub} := 42 \]
Dynamic Info Flow Tracking

- Flow-Insensitive:

\[ \Gamma(\text{sec}>0) = H \]

\[ pc \mapsto L \]

\[ \text{sec} \mapsto H \]

\[ \text{pub} \mapsto L \]

```plaintext
if (sec>0) then
  sec := 42
else
  skip;
pub := 0
```
Dynamic Info Flow Tracking

- Flow-Insensitive:

\[
\begin{align*}
\Gamma(\text{sec}>0) &= H \\
\text{pc} &\mapsto H \sqcup L \\
\text{sec} &\mapsto H \\
\text{pub} &\mapsto L
\end{align*}
\]

```
if (sec>0) then
  sec := 42
else
  skip;
pub := 0
```
Dynamic Info Flow Tracking

- Flow-Insensitive:

\[ pc \sqcup \Gamma(42) \sqsubseteq \Gamma(\text{sec}) \]
\[ (H \sqcup L) \sqcup L \sqsubseteq H \]

\[
\begin{align*}
if \ (\text{sec}>0) & \text{ then} \\
\text{sec} & := 42 \\
\text{else} & \\
\text{skip;} & \\
\text{pub} & := 0
\end{align*}
\]

\[ pc \mapsto H \sqcup L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
Dynamic Info Flow Tracking

- Flow-Insensitive:

```plaintext
if (sec>0) then
    sec := 42
else
    skip;
pub := 0
```

\[
\begin{align*}
p_c \sqcup \Gamma(0) & \sqsubseteq \Gamma(\text{pub}) \\
L \sqcup L & \sqsubseteq L
\end{align*}
\]
Dynamic Info Flow Tracking

- Flow-Insensitive:

\[ pc \sqcup \Gamma(\text{sec}>0) = H \]

\[ pc \rightarrow L \]

\[ \text{sec} \rightarrow H \]

\[ \text{pub} \rightarrow L \]

if (sec>0) then
  pub := 42
else
  skip
Dynamic Info Flow Tracking

- Flow-Insensitive:

\[
pc \sqcup \Gamma(\text{sec} > 0) = H \quad pc \mapsto H \sqcup L \\
sec \mapsto H \\
pub \mapsto L
\]

if (sec > 0) then
  pub := 42
else
  skip
Dynamic Info Flow Tracking

• Flow-Insensitive:

\[
\begin{align*}
pc \cup \Gamma(42) &\subseteq \Gamma(\text{pub}) \\
(H \cup L) \cup L &\subseteq L
\end{align*}
\]

\[
\begin{align*}
pc \mapsto H \cup L \\
\text{sec} &\mapsto H \\
\text{pub} &\mapsto L
\end{align*}
\]

if (sec > 0) then

\[
\text{pub} := 42
\]

else

\[
\text{skip}
\]
Flow-Sensitive Dynamic

- Natural thing to do is allow the security context to be **flow sensitive**
  - i.e., the mapping from vars to security levels can change during execution
  - (Can do a similar thing with a flow-sensitive type system)
- Accepts more programs!
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(\text{sec}) = H \]

\[ pc \mapsto L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto L \]

\[
\begin{align*}
x & := \text{sec}; \\
x & := 0; \\
\text{output sec to L}
\end{align*}
\]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(\text{sec}) = H \]

\[ pc \mapsto L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto H \]

\[ x := \text{sec}; \]
\[ x := 0; \]
\[ \text{output sec to L} \]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(0) = L \]

\[ pc \mapsto L \]
\[ sec \mapsto H \]
\[ pub \mapsto L \]
\[ x \mapsto H \]

\[ x := \text{sec}; \]
\[ x := 0; \]
\[ \text{output sec to L} \]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(0) = L \]

\[
\begin{align*}
pc & \mapsto L \\
\sec & \mapsto H \\
\text{pub} & \mapsto L \\
x & \mapsto L
\end{align*}
\]

\[
\begin{align*}
x := & \text{sec;} \\
x := & \text{0;} \\
\text{output } & \text{sec to } L
\end{align*}
\]
Flow-Sensitive Dynamic

\[ pc \cup \Gamma(\text{sec}) \subseteq L \]
\[ L \cup H \subseteq L \]

\[ pc \mapsto L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto L \]

\[ x := \text{sec}; \]
\[ x := 0; \]
\[ \text{output sec to } L \]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(sec > 0) = H \]

\[
\begin{align*}
pc &\mapsto L \\
sec &\mapsto H \\
pub &\mapsto L \\
x &\mapsto L \\
\end{align*}
\]

\[
\text{if (sec > 0)} \\
x := 1 \\
\text{else} \\
\text{skip;} \\
\text{output x to L}
\]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(\text{sec}>0) = H \]

\[
\begin{align*}
\text{sec} &\mapsto H \\
\text{pub} &\mapsto L \\
x &\mapsto L
\end{align*}
\]

if (sec > 0)
  x := 1
else
  skip;
output x to L
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(0) = H \]

\[
\begin{align*}
pc &\mapsto H \sqcup L \\
sec &\mapsto H \\
\text{pub} &\mapsto L \\
x &\mapsto L
\end{align*}
\]

if (sec > 0)
\[
\begin{align*}
x &:= 1 \\
\text{else} \\
\text{skip;}
\end{align*}
\]

output x to L
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(0) = H \]

\[ pc \mapsto H \sqcup L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto H \]

if (sec > 0)
  x := 1
else
  skip;
output x to L
Flow-Sensitive Dynamic

\[ pc \cup \Gamma(x) \subseteq L \]
\[ L \cup H \subseteq L \]

\[ p \mapsto L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto H \]

\[ \text{if (sec > 0)} \]
\[ \text{x := 1} \]
\[ \text{else} \]
\[ \text{skip;} \]
\[ \text{output x to L} \]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(\text{sec} > 0) = H \]

\[ pc \mapsto L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto L \]

\[ \text{if (sec} > 0) \]
\[ x := 1 \]
\[ \text{else} \]
\[ \text{skip} \; ; \]
\[ \text{output x to L} \]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(\text{sec}>0) = H \]

\[ pc \mapsto H \sqcup L \]
\[ \text{sec} \mapsto H \]
\[ \text{pub} \mapsto L \]
\[ x \mapsto L \]

\[ \text{if } (\text{sec} > 0) \]
\[ x := 1 \]
\[ \text{else} \]
\[ \text{skip;} \]
\[ \text{output } x \text{ to } L \]
Flow-Sensitive Dynamic

\[ pc \mapsto H \sqcup L \]
\[ sec \mapsto H \]
\[ pub \mapsto L \]
\[ x \mapsto L \]

if \( (sec > 0) \)
  \( x := 1 \)
else
  skip;
output \( x \) to \( L \)
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(x) \subseteq \Gamma(L) \]

- This is an implicit (aka indirect) flow!
- If we allow it, on some executions we will leak information.
  - So called “half-bit” leak.
  - Can combine 2 “half-bit” leaks to reliably leak a bit!

\[ pc \to L \]
\[ sec \to H \]
\[ pub \to L \]
\[ x \to L \]

\[
\begin{align*}
\text{if (sec > 0)} & \\
& \text{x := 1} \\
\text{else} & \\
& \text{skip;} \\
& \text{output x to L}
\end{align*}
\]
Flow-Sensitive Dynamic

\[ pc \sqcup \Gamma(x) \subseteq \Gamma(L) \]

- This is an implicit (aka indirect) flow!
- If we allow it, on some executions we will leak information.
  - So called “half-bit” leak.
  - Can combine 2 “half-bit” leaks to reliably leak a bit!

```plaintext
x := 0; y := 1;
if (sec > 0)
x := 1
else
  skip;
if (x = 1)
skip
else
  y := 0
output y to L
```
No-Sensitive Upgrade

• Austin and Flanagan (2009)
• Don’t raise level of variables when \( pc \) is high
  • i.e., only raise level of variable \( x \) if currently \( pc \sqsubseteq \Gamma(x) \)
• Some slightly more permissive variations are possible
Dynamic vs Static

- **Flow-insensitive** dynamic tracking can be more precise (for termination-insensitive NI) than flow-insensitive type system
- **Flow-sensitive** dynamic tracking and flow-sensitive type system are incomparable (for termination-insensitive NI)
- **Hybrid systems** combine static and dynamic techniques

![Diagram](image)

Figure 2. Relation between programs accepted by type systems and monitors
Other Fine-Grained Enforcement Mechanisms

- Dataflow analyses
- Abstract interpretation
- Program dependence graphs/program slicing
- Program rewriting
- Symbolic execution
- Relational program logics
- ...

Stephen Chong, Harvard University
**Coarse-Grain Info Flow Control**

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)
• **Computation containers** track what information comes into container
  • Think process, or maybe object
  • Maintain a **high-water mark**: highest security level seen
  • All info in container is treated as potentially tainted with high water mark
• Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)

![Diagram showing different security levels and information flow]

Level: L
- H
- 42

Level: M
- 23
- 19

Level: H
- 42
Selected References


Beyond Confidentiality
Confidentiality and Integrity

- So far, we have considered information flow for confidential information
- We can also think about information flow for integrity
Confidentiality and Integrity

• For confidentiality: we want to restrict flow of secret data
• For integrity: we want to restrict flow of untrusted data
Confidentiality and Integrity

• For confidentiality: we want to restrict flow of secret data
• For integrity: we want to restrict flow of untrusted data
Noninterference

• The semantic condition is exactly the same!
• The duality between confidentiality and integrity is the direction of “trust” in the lattice

Definition: Program c is noninterfering if:
For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2, \ell$
if $\sigma_1 = \ell \sigma_2$ and $\langle c, \sigma_1 \rangle \Downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \Downarrow \sigma'_2$
then $\sigma'_1 = \ell \sigma'_2$
However...

• There are differences between confidentiality and integrity

• Code
  • Many well-principled mechanisms for the integrity of code
    • Code signing
    • Checking of mobile code (bytecode verification, proof-carrying code, type checking, ...)
    • Sandboxing
  • Not so for confidentiality
    • There are impossibility results about the confidentiality of code...
More Differences

• Termination, timing, power consumption, and other side channels
  • Maybe less severe...
    • Do we care if the attacker can affect the acoustic emanations of a CPU?
  • Some covert channel attacks become availability attacks, resource consumption attacks
Reclassification

• The dual of declassification is called endorsement
  • Declassification: making information less confidential
  • Endorsement: making information more trusted
• Both move information downwards in the lattice
Endorsement

• Aspects of declassification apply to endorsement
  • *What* information is being endorsed?
  • *Who* is responsible for endorsing it? *Who* receives the endorsed information?
  • *Where* in the system (or info-flow lattice) does endorsement happen?
  • *When* is information endorsed?
• Quantitative information flow: *how much* information is leaked
  • Contamination vs suppression (Clarkson & Schneider)
  • Contamination = how much untrusted input contaminates trusted output
    • Dual for confidentiality: how much secret input present in public output
  • Suppression = how much trusted input is suppressed in trusted output
    • No confidentiality dual!
Combining Confidentiality and Integrity

- Given a lattice for confidentiality $(\Lambda_C, \sqsubseteq_C)$ and a lattice for integrity $(\Lambda_I, \sqsubseteq_I)$, we can combine them into a single lattice $(\Lambda, \sqsubseteq)$ where
  
  - $\Lambda = \Lambda_C \times \Lambda_I = \{ (\ell_c, \ell_i) \mid \ell_c \in \Lambda_C, \ell_i \in \Lambda_I \}$
  
  - $(\ell_c, \ell_i) \sqsubseteq (\ell'_c, \ell'_i)$ iff $\ell_c \sqsubseteq_C \ell'_c$ and $\ell_i \sqsubseteq_I \ell'_i$
Combining Confidentiality and Integrity

**Confidentiality Levels**
Who can *read* information?
E.g., in A,B, Alice can read it, and Bob can read it (Charlie can not)

**Integrity Levels**
Who can *write* information?
E.g., in A,B, Alice can write it, and Bob can write it (Charlie can not)
Interaction Between Confidentiality and Integrity

- Consider a program that declassifies some data

```
secret1 := ...;
secret2 := ...;
x := secret1;
pub = declassify(x)
```

- Suppose the attacker can influence which secret is declassified

```
secret1 := ...;
secret2 := ...;
if (low_input) then x := secret1
else x := secret2
pub = declassify(x)
```

- Attacker can cause the wrong data to be declassified
  - So-called “laundering attack”
Robust Declassification

• Zdancewic and Myers (2001)
• Intuitive idea: an active attacker should not learn more than a passive attacker
  • Active attacker: providing low-integrity inputs
  • Passive attacker: just observing
• This implies that the data to declassify, and the decision to declassify it, should be high integrity
Typing Rule for Robust Declassification

• Rule for assignment

\[
\frac{\Gamma \vdash e : \tau_{\ell_e} \quad \ell_e \sqcup pc \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e} \quad \Gamma(x) = \tau_{\ell_x}
\]

• Equivalent rule for assignment

\[
\frac{\Gamma \vdash e : \tau_{\ell_e} \quad pc \sqsubseteq \ell_x \quad \ell_e \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e} \quad \Gamma(x) = \tau_{\ell_x}
\]
Typing Rule for Robust Declassification

- Equivalent rule for assignment

\[
\Gamma 
\vdash e : \tau_{\ell_e} \quad \text{pc} \sqsubseteq \ell_x \quad \ell_e \sqsubseteq \ell_x \quad \frac{\Gamma, \text{pc} \vdash x := e}{\Gamma, \text{pc} \vdash x := e} \quad \Gamma(x) = \tau_{\ell_x}
\]

- Rule for declassification

\[
\Gamma \vdash e : \tau_{\ell_{from}} \quad \text{pc} \sqsubseteq \ell_{to} \quad \text{pc} \sqsubseteq (\text{Secret, Trusted}) \quad \ell_{from} \sqsubseteq (\text{Secret, Trusted}) \quad \text{integOf}(\ell_{from}) = \text{integOf}(\ell_{to}) \quad \frac{\Gamma, \text{pc} \vdash x := \text{declassify}(e)}{\Gamma, \text{pc} \vdash x := \text{declassify}(e)} \quad \Gamma(x) = \tau_{\ell_{to}}
\]

- Decision to declassify is trusted

- It is declassification only, not endorsement

- Secret, Untrusted
  - Public, Untrusted
  - Secret, Trusted
    - Public, Trusted
Typing Rule for Robust Declassification

\[
\begin{align*}
\Gamma \vdash e : & \tau_{\ell_{\text{from}}} \\
pc & \sqsubseteq \ell_{\text{to}} \\
pc & \sqsubseteq (\text{Secret, Trusted}) \\
\ell_{\text{from}} & \sqsubseteq (\text{Secret, Trusted}) \\
\text{integOf}(\ell_{\text{from}}) & = \text{integOf}(\ell_{\text{to}}) \\
\end{align*}
\]

\[
\Gamma, pc \vdash x := \text{declassify}(e)
\]

- Intuition: for any principal \( p \), if the declassification lets \( p \) read the data, \( p \) should not have influenced it
  - \( \forall p. \ p \in \text{readers}(\ell_{\text{to}}) - \text{readers}(\ell_{\text{from}}) \Rightarrow p \notin \text{writers}(\ell_{\text{from}}) \)
  - \( \forall p. \ p \in \text{readers}(\ell_{\text{to}}) \Rightarrow p \in \text{readers}(\ell_{\text{from}}) \text{ or } p \notin \text{writers}(\ell_{\text{from}}) \)
  - \( \text{readers}(\ell_{\text{from}}) \supseteq \text{readers}(\ell_{\text{to}}) \cap \text{writers}(\ell_{\text{from}}) \)
  - \( \text{readers}(\ell_{\text{from}}) \sqsubseteq \text{readers}(\ell_{\text{to}}) \sqcup \text{writers}(\ell_{\text{from}}) \)
  - \( \ell_{\text{from}} \sqsubseteq \ell_{\text{to}} \sqcup \text{writersToReaders}(\ell_{\text{from}}) \)
Typing Rule for Robust Declassification

\[ \Gamma \vdash e : \tau_{\ell_{\text{from}}} \quad pc \subseteq \ell_{\text{to}} \quad pc \subseteq (\text{Secret, Trusted}) \quad \ell_{\text{from}} \subseteq (\text{Secret, Trusted}) \quad \text{integOf}(\ell_{\text{from}}) = \text{integOf}(\ell_{\text{to}}) \]

\[ \Gamma, pc \vdash x := \text{declassify}(e) \]

\[ \Gamma \vdash e : \tau_{\ell_{\text{from}}} \quad pc \subseteq \ell_{\text{to}} \quad \text{integOf}(\ell_{\text{from}}) = \text{integOf}(\ell_{\text{to}}) \quad \ell_{\text{from}} \subseteq \ell_{\text{to}} \cup \text{writersToReaders}(\ell_{\text{from}}) \quad \ell_{\text{from}} \subseteq \ell_{\text{to}} \cup \text{writersToReaders}(pc) \]

\[ \Gamma, pc \vdash x := \text{declassify}(e) \]
What About Endorsement?

- Equivalent of robust declassification for integrity is **transparent endorsement** (Cecchetti et al., 2017)
- Intuitively: data and decision to endorse should be public
- Nonmalleable info flow = robust declassification + transparent endorsement
Dependency

• At its core, noninterference is about (in)dependency
• Techniques for noninterference are also good for dependency
• E.g.,
  • Binding-time analysis, slicing, ... (Abadi et al. 1999)
  • Tracking and restricting errors in computation (Sampson et al. 2011)
Selected References


