463.7 Information Flow

Computer Security II
CS463/ECE424
University of Illinois
Downloadable financial planner:

- Access control insufficient
- Encryption necessary, but also insufficient
Noninterference

- Private data does not *interfere* with network communication
- Baseline confidentiality policy
Example: Payment Smart Card

- Smart card for financial transactions
- Card contains sensitive financial information used to make a transaction
- Used in the terminal of a merchant
- Card is tamper-resistant, protects card secrets from both the merchant and card holder
- Java cards provide limited card-based computation
- Must split the computation between the card, the terminal, and network connections
Model of Noninterference

- Represent noninterference as a relation between groups of users and commands
- Users in group G do not interfere with those in group G’ if the state seen by G’ is not affected by the commands executed by members of G
State Automaton

- U – Users
- S – States
- C – Commands
- Out – Outputs
- \(\text{do} : S \times U \times C \rightarrow S\) – state transition function
- \(\text{out} : S \times U \rightarrow \text{Out}\) – output function
- \(s_0\) – initial machine state
Capability System

- U, S, Out – users, states, commands, and outputs as before (in a state machine)
- SC – State commands
- Capt – Capability tables
- CC – Capability commands
- out : S × Capt × U → Out
- do : S × Capt × U × SC → S
- cdo : Capt × U × CC → Capt – Capability selection function
- s₀ ∈ S and t₀ ∈ Capt – Initial state and capability tables
Transition Function

• $C = SC \cup CC$ - Commands

• $csdo : S \times Capt \times U \times C \rightarrow S \times Capt$
  - $csdo(s,t,u,c) = (do(s,t,u,c),t)$ if $c \in SC$
  - $csdo(s,t,u,c) = (s,cdo(s,t,u,c))$ if $c \in CC$

• $csdo^* : S \times Capt \times (U \times C)^* \rightarrow S \times Capt$
  - $csdo^*(s,t,nil) = (s,t)$
  - $csdo^*(s,t,w.(u,c)) = csdo(csdo^*(s,t,w),u,c)$

• $[[w]] = csdo^*(s_0,t_0,w)$

• $[[w]]_u = out([[w]],u)$
• Let $G \subseteq U$ and $A \subseteq C$ and $w \in (U \times C)^*$
• $P_G(w) = \text{subsequence of } w \text{ obtained by eliminating pairs } (u,c) \text{ where } u \in G$
• $P_A(w) = \text{subsequence of } w \text{ obtained by eliminating pairs } (u,c) \text{ where } c \in A$
• $P_{G,A}(w) = \text{subsequence of } w \text{ obtained by eliminating pairs } (u,c) \text{ where } u \in G \text{ and } c \in A$
Noninterference

- M state machine and \( G, G' \subseteq U \) and \( A \subseteq C \)
- \( G : | G' \) iff \( \forall w \in (U \times C)^*. \forall u \in G'. \quad [[w]]_u = [[p_G(w)]]_u \)
- \( A : | G \) iff \( \forall w \in (U \times C)^*. \forall u \in G. \quad [[w]]_u = [[p_A(w)]]_u \)
- \( A,G : | G' \) iff \( \forall w \in (U \times C)^*. \forall u \in G'. \quad [[w]]_u = [[p_{A,G}(w)]]_u \)
• **Noninterference assertions** have the forms
  
  \[ G : | G' \]
  \[ A : | G \]
  \[ A,G : | G' \]

• A *security policy* is a set of noninterference assertions
Example 1

• A : | {u}
• The commands in A do not interfere with the state of user u
Example 2 MLS

- **Level**: $U \rightarrow L$ - assignment of security levels in $L$
- **Above($\lambda$)** = $\{ u \in U \mid \lambda \sqsubseteq \text{Level}(u) \}$
- **Below($\lambda$)** = $\{ u \in U \mid \text{Level}(u) \sqsubseteq \lambda \}$
- $M$ is *multi-level secure* with respect to $L$ if, for all $\lambda \sqsubseteq \lambda'$ in $L$, $\text{Above}(\lambda') : | \text{Below}(\lambda)$
• G is invisible if $G :| G^c$ where $G^c$ is the complement of G in U

• **Proposition 1:** If $M,L$ is multi-level secure, then $\text{Above}(\lambda)$ is invisible for every $\lambda \in L$. 
Example 4 Isolation

- A group of users $G$ is *isolated* if: $G : | G^c$ and $G^c : | G$.
- A system is *completely* isolated if every user in $U$ is isolated.
Example 5 Channel Control

• View a *channel* as a set of commands $A$

• We can assert that groups of users $G$ and $G'$ can only communicate through channel $A$ with the following two noninterference assertions:

\[ \text{A}^c,G :| G' \]
\[ \text{A}^c,G' :| G \]
Example 6 Information Flow

\[ A, A_1, A_2 \]

\[ u', u_1, u_2 :| u \]
\[ u_1, u_2 :| u' \]
\[ u_1 :| u_2 \]
\[ u_2 :| u_1 \]
\[ A^c, u :| \{ u', u_1, u_2 \} \]
\[ A_1^c, u' :| \{ u_1 \} \]
\[ A_2^c, u' :| \{ u_2 \} \]
Example 7 Security Officer

• Let $A$ be the set of commands that can change the security policy

• $\text{seco} \in U$ is the only individual permitted to use these commands to make changes

• This is expressed by the following policy: $A, \{\text{seco}\}^c : | U$
It is possible to analyze information flows in programs with an information theory foundation.

Intuition: info flows from x to y as a result of a sequence of commands c if you can deduce information about x before c from the value in y after c.
Example 1

- $y := x$
  - If we learn $y$, then we know $x$
  - Clearly information flows from $x$ to $y$
Example 2

- Suppose we are given
  
r := x
  
r := r - r
  
y := 1 + r

- Does information flow from x to y?
- It does not because r = 0 after the second command.
Consider this branching command:

```plaintext
if x = 1 then y := 0 else y := 1;
```

If we find after this command that \( y \) is 0, then we know that \( x \) was 1.

So information flowed from \( x \) to \( y \).
Implicit Flow of Information

• Information flows from \(x\) to \(y\) without an *explicit* assignment of the form \(y := f(x)\) where \(f(x)\) an arithmetic expression with variable \(x\)

• Recall the example from previous slide:
  \[
  \text{if } x = 1 \text{ then } y := 0 \text{ else } y := 1;
  \]

• So we must look for *implicit* flows of information to analyze program
Conservative Automated Analysis of Flow

• Example 2 depends on an arithmetic property of subtraction
• It is impossible to take each such property into account when doing an automated analysis
  – Ultimately undecidable
• Hence an automated analysis will be a conservative approximation of information flows
  – All flows can be found (even if trivially!)
  – Some non-flows (false positives) will be found
Compiler-Based Mechanisms

- Detect unauthorized information flows in a program during compilation
- Analysis not precise, but secure
  - If a flow *could* violate policy (but may not), it is unauthorized
  - No unauthorized path along which information could flow remains undetected
- Set of statements *certified* with respect to information flow policy if flows in set of statements do not violate that policy
Example

```c
if x = 1 then y := a else y := b;
```

- Info flows from \( x \) and \( a \) to \( y \), or from \( x \) and \( b \) to \( y \)
- Certified only if information from the security class \( x \) of \( x \) is allowed to flow into the security class \( y \) of \( y \) and similar conditions hold for \( a \) and \( b \) relative to \( y \).
- Write: \( x \leq y \) and \( a \leq y \) and \( b \leq y \)
  - Note flows for both branches must be true unless compiler can determine that one branch will never be taken
Declarations

x: int class \{A, B\}

- Means x is an integer variable with security class at least lub\{ A, B \} so lub\{ A, B \} \leq x.
- Basic case is two security classes, High and Low.
Assignment Statements

\[ x := y + z; \]

- Information flows from \( y, z \) to \( x \), so this requires
  \[ \text{lub}\{ y, z \} \leq x \]

More generally:

\[ y := f(x_1, \ldots, x_n) \]

- Require \( \text{lub}\{ x_1, \ldots, x_n \} \leq y \)
\[ x := y + z; \quad a := b \times c - x; \]

- First statement: \( \text{lub}\{ y, z \} \leq x \)
- Second statement: \( \text{lub}\{ b, c, x \} \leq a \)

So, both must hold (i.e., be secure)

More generally:

\[ S_1; \ldots; S_n; \]

- Each individual \( S_i \) must be secure
while $i < n$ do
begin $a[i] := b[i]; i := i + 1; \text{end}$

• Same ideas as for "if", but must terminate

More generally:
while $f(x_1, \ldots, x_n)$ do $S$

• $S$ must be secure
• lub$\{x_1, \ldots, x_n\} \leq$ glb$\{y \mid y \text{ target of an assignment in } S\}$
• Loop must terminate
Conditional Statements

if $x + y < z$
then $a := b$
else $d := b \times c - x$; end

• The statement executed reveals information about $x$, $y$, $z$, so $\text{lub\{x, y, z\}} \leq \text{glb\{a, d\}}$

More generally:
if $f(x_1, \ldots, x_n)$ then $S_1$ else $S_2$; end

• $S_1$, $S_2$ must be secure
• $\text{lub\{x_1, \ldots, x_n\}} \leq \text{glb\{y | y target of assignment in S_1, S_2\}}$
begin
  i,n: integer security class L;
  flag: Boolean security class L;
  f1,f2: file security class L;
  x,sum: integer security class H;
  f3,f4: file security class H;
begin
  i := 1;
  n := 0;
  sum := 0;
  while i ≤ 100 do
    begin
      input flag from f1;
      output flag to f2;
      input x from f3;
      if flag then
        begin
          n := n + 1;
          sum := sum + x
        end;
      i := i + 1
    end;
    output n, sum, sum/n to f4
end
end
Need to Handle More

- Procedures
- Arrays
- Goto Statements
- Exceptions
- Infinite loops
- Concurrency
- Etc
Reading

  – Chapter 8 up to the beginning of 8.2.1.
  – Chapter 16 sections 16.1 and 16.3
Is the G/M notion of interference too strong? Consider the following two case studies.
## Audit

Consider the security officer in example 7. Shouldn’t the officer see audit information from the users who attempt to execute security commands?

## Secret Communication

A general tells his army that if they see a green flag they should attack from the left but if they see a red flag they should attack from the right. The general raises the green flag and the enemy forces see this. Did the signal “interfere” with the enemy?