Topic 17: Non-interference and Non-deducibility
Optional Readings for This Lecture


• Non-deducibility is from the paper “A Model of Information” by David Sutherland
  • Not available online
What is a Security Model?

• A model describes the system
  – e.g., a high level specification or an abstract machine description of what the system does
• A security policy
  – defines the security requirements for a given system
• Verification techniques that can be used to show that a policy is satisfied by a system
• System Model + Security Policy = Security Model
Motivations

• Multi-level security is about information flow
  – Information in high level objects should not flow into low-level subjects

• The BLP model describes access control mechanisms that prevents illegal information flow, but not the meaning of no illegal information flow
  – BLP describes “how”, not “what” for information flow protection
    • E.g., define secure encryption by giving a particular encryption algorithm and say this is secure encryption
  – As a result, BLP does not prevent information flow through covert channels
  – Also, it doesn’t say whether other mechanisms can be used do information flow protection
Non-interference in Programs

- Consider the following functions, is there information flow between x and output of the functions?

```c
add(int x, int y) {
    return x+y;
}

check_pw(char *s) {
    char *x;
    return strcmp(x,s);
}

g(int x, int y){
    return x*y/x;
}

d(int x, int y) {
    if x>0 return y+y;
    else return 2*y;
}
```
Deterministic Non-Interference in Programs

• A set $X$ of inputs is non-interfering with a set $Y$ of outputs if and only if
  – No matter what values $X$ take, the outputs $Y$ remain the same
    • When one changes only values of inputs in $X$, the output remain unchanged
    • Observing only $Y$, one learns nothing about any input in $X$.
  – More formally, let $Y = f(X, Z)$, where $f$ is a deterministic function, and $X, Z$ represents two sets of inputs, $X$ is non-interfering with $Y$ iff
    \[ \forall Z_0 \exists Y_0 \forall X_0 \ f(X_0, Z_0) = Y_0 \]
    or equivalently,
    \[ \forall Z_0 \forall X_0 \forall X_1 \ f(X_0, Z_0) = f(X_1, Z_0) \]
  – $X$ interferes with $Y$ iff.
    \[ \exists Z_0 \exists X_0 \exists X_1 \ f(X_0, Z_0) \neq f(X_1, Z_0) \]
• For randomized programs, non-interference is harder to define, and we do not cover it in this course
More on Non-interference Properties

- Two classes of techniques to ensure that security properties are satisfied by programs
  - Monitor execution of a program and deny illegal actions or terminate the program if illegal action is detected.
    - Can enforce BLP property.
    - Cannot enforce non-interference.
      - Why? Because non-interference is not defined on one execution of a program; it is a property on a program’s behaviors on different inputs.
  - Statically verifying that certain non-interference relation holds by analyzing the program
    - Can be used only with access to source code
Language-Based Security

- Using programming language technique to ensure certain security properties hold
  - A large body of work focuses on using type theory and compiling-time checks to ensure information-flow properties

- Challenges to apply in real world:
  - Non-interference is often too strong
    • Suppose that one want to ensure that a secret password is not leaked, can one require non-interference between the password input and observable output?
    • Needs declassification mechanism that specify certain information dependent on sensitive inputs can be leaked.
  - Specifying such policies is impractical
    • Too much work for programmers, especially for large programs
    • Many policies need to be determined by end users, not programmers
  - Need source code, unable to deal with the real security challenge of external code.
The Non-Interference Model in the Original Goguen-Meseguer paper

- A state-transition model, where state changes occur by subjects executing commands
  - $S$: set of states
  - $U$: set of subjects
  - $SC$: set of state commands
  - $Out$: set of all possible outputs
  - $do: S \times U \times SC \rightarrow S$
    - $do(s,u,c)=s'$ means that at state $s$, when $u$ performs command $c$, the resulting state is $s'$
  - $out: S \times U \rightarrow Out$
    - $out(s,u)$ gives the output that $u$ sees at state $s$
  - $s_0 \in S$ initial state

Model focuses on interfaces (inputs/outputs) of a system, rather than internal aspects (e.g., objects)
Security Policies in the Non-interference Model

- A security policy is a set of noninterference assertions
- Definition of noninterference: Given two group of users \( G \) and \( G' \), we say \( G \) does not interfere with \( G' \) if for any sequence of commands \( w \),
  - \( \text{View}_{G'}(w) = \text{View}_{G'}(P_G(w)) \)
    - \( P_G(w) \) is \( w \) with commands initiated by users in \( G \) removed.
    - No matter what users in \( G \) do, users in \( G' \) will observe the same.
- Implicit assumptions:
  - Initial state of the system does not contain any sensitive information
  - Information comes into the system by commands
  - Only way to get information is through outputs
Comparisons of the BLP work & the Noninterference work

• Differences in model
  – BLP models internals of a system (e.g., objects)
  – GM models the interface (input & output)

• Differences in formulating security policies
  – BLP specifies access control requirement, noninterference specifies information flow goal

• Noninterference could address covert channels concerns
  – Provided that one defines observable behavior to include those in covert channels; doesn’t make stopping covert channel easier

• Under noninterference, a low user is allowed to copy one high-level file to another high-level file
  – In general not allowed by BLP
Evaluation of The Non-Interference Policy

- The notion of noninterference is elegant and natural
  - Focuses on policy objective, rather than mechanism, such as BLP
  - Could be useful in other settings
- Mostly concerned with deterministic systems
  - For randomized or otherwise non-deterministic systems, definition is more complicated
- May be too restrictive
  - e.g., consider encrypt and then communicate
Non-deducibility

- Attempt to define information flow in non-deterministic as well as deterministic systems
- Intuition: there is no information flow between X and Y, iff., when observing only Y, one can never eliminate any value from the domain in X as a possible value
- Definition: let \( Y = f(X, Z) \), where \( f \) is not necessarily deterministic, there is information flow between X and Y in the non-deducibility sense iff.

  \[
  \exists Y_0 \in \{ f(X, Z) \} \quad \exists X_0 \text{ s.t. } Y_0 \notin \{ f(X_0, Z) \}
  \]

  - When one observes the value of Y is \( Y_0 \), one learns that \( X \neq X_0 \).
  - There is no information flow between X and Y in the non-deducibility sense when \( \forall Y_0 \in \{ f(X, Z) \} \quad \forall X_0 \exists Z_0 \text{ s.t. } Y_0 \in \{ f(X_0, Z_0) \} \)

- Go to the examples for non-interference
An Example Illustrating that Non-deducibility is Too Weak

• A high user and a low user
  – the high user can write to a file
    • one letter at a time
  – the low user can try to read the n’th character in a file
    • if file is shorter than n, or if the n’th character is blank, returns a random letter
    • otherwise, with 99.9% probability return the letter, and with 0.1% probability return a random letter

• The system is nondeducible secure
• The system is intuitively insecure
• Non-deducibility can often be too weak. It deals with possibilistic inference, not probabilistic inference
Examples:

High int x = …;
High int y = …;
Low int z;
if x>0 z = y+y;
else z = 3*y;
• x interferes with z
• y interferes with z
• x and z are not non-deduciable secure
• y and z are not non-deduciable secure

High int x = …;
High int y = …;
Low int z;
if x>0 z = y+y;
else z = 2*y;
• x does not interfere with z
• y interferes with z
• x and z are non-deduciable secure
• y and z are not non-deduciable secure
Examples

High int x = ...;
High int y = ...;
Low int z1 = x + y;
Low int z2 = x − y;

• x interferes with z1
• x interferes with z2
• x and z1 are non-deduciable secure
• x and {z1,z2} are not non-deduciable secure

High char * x = ...;
Low char * entered_pw = ...;
Low boolean z;
z = strcmp(entered_pw,x);

• x interferes with z
• x and {z, entered_pw} are not non-deduciable secure
Relationships Between Nondeducibility & Noninterference

• For deterministic systems with just one high input var (and possibly many other low input vars) and one low output, a system is noninterference secure if and only if it is nondeducibility secure.

• For deterministic systems with more than one high input vars, non-interference is stronger than non-deducibility.
Proof.

- Theorem: For deterministic programs with just one high input variable x, let Z be the set of all low variables, x does not interfere with the set Z if and only if x and Z are nondeducible secure.

- Proof. If x does not interfere with Z, no matter what values x takes, the variables in Z are uniquely determined by inputs in Z. Observing values in Z cannot eliminate any value for x.

- If x interferes with Z, then there exist \( x_1 \neq x_2 \) and \( Z_2 \neq Z_1 \) such that \( Z = Z_1 \) when \( x = x_1 \) and \( Z = Z_2 \neq Z_1 \) when \( x = x_2 \). Observing \( Z = Z_2 \), one knows \( x \neq x_1 \), making x and X not nondeducible secure. This is because as x is the only high var and the system is deterministic, when fixing input variables in Z to values in \( Z_2 \), the output variables are fixed as well.
Relationship Between Security Notions

- Perfect secrecy
- IND-CPA security
- Non-interference
- Non-deducability
Next Lecture

- Data Privacy