Advanced Systems Security: Program Information
Flow Control

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Problem

• A program is trusted to enforce a system’s policy
  ‣ How do we know?
• So what can we do?
Problem

Alice - X Window

Remote Client A

Remote Client B
Problem
Problem

Secret

Alice - XWindow

Remote Client A

Remote Client B
Problem

Alice - X Window

Remote Client A

Remote Client B

Secret
Problem

Alice - X Window

Remote Client A

Secret

Remote Client B

Secret

STOP
Problem

- Remote Client A
  - Alice - X Window
  - Remote Client B
What’s a Program?

- Program parts
  - Statements (Expressions), Variables, Control Statements, Procedures, Arguments, System calls/Library calls

- What does a program look like from a security perspective?
  - Variables have data (may have secrecy/integrity reqs)
  - Variable values may come from external sources
  - Variable values may be assigned to one another
  - Variables may be written out of the program (sink)
What’s a Program?

• Ensure that secret data is encrypted before it is released.

1. user_name = getString();
2. secret_data_1 := getPasswdFromUser();
3. secret_data_2 := getPasswdFromUser();
4. If(secret_data_1 == secret_data_2)
   5.   writeToFile(secret_data_1);
   6. else
   7.   writeToOutput(“Passwords do not match”);
What’s a Program?

- Ensure that secret data is encrypted before it is released.

1. `user_name = getString();`
2. `secret_data_1 := getPasswdFromUser();`
3. `secret_data_2 := getPasswdFromUser();`
4. `If(secret_data_1 == secret_data_2)`
5. `writeToFile(encrypt(secret_data_1));`
6. `else`
7. `writeToOutput(“Passwords do not match”);`
It’s the Data Flow!!

- Data input to a program may have security requirements
  - E.g., it is secret

- The program statements enable the data to “flow” through the program
  - Track each variable’s label (based on the data it’s seen)

- Enforce a data security requirements on information flows
  - Can that data be sent out to a file?

- Can connect OS/VM and program enforcement
Concepts

- Attach **security labels** to program data
- Enable static checking of information flows
  - Compatible with Denning’s model
  - Only a program with legal information flows will compile
- Programmers can *declassify* labels
  - Upgrade integrity
  - Downgrade secrecy
- Generalize approach
  - Label polymorphism
  - Run-time label checking
Denning’s Lattice Model

- Formalizes information flow models
  - $FM = \{N, P, SC, /, >\}$
- Shows that the information flow model instances form a lattice
  - $N$ are objects, $P$ are processes,
  - $\{SC, >\}$ is a partial ordered set,
  - $SC$, the set of security classes is finite,
  - $SC$ has a lower bound,
  - and $/$ is a lub operator
- Implicit and explicit information flows
- Semantics for verifying that a configuration is secure
- Static and dynamic binding considered
- Biba and BLP are among the simplest models of this type
Implicit and explicit flows

- **Explicit**
  - Direct transfer to $b$ from $a$ (e.g., $b = a$)

- **Implicit**
  - Where value of $b$ may depend on value of $a$ indirectly (e.g., if $a = 0$, then $b = c$)

- **Model covers all programs**
  - Statement $S$
  - Sequence $S_1$, $S_2$
  - Conditional $c$: $S_1$, ..., $S_m$

- **Implicit flows only occur in conditionals**
Preventing Implicit Flows

- Hard to do without static analysis
- Consider code fragment
  
  \[
  \begin{align*}
  x &:= 0 \\
  \text{if } b \text{ then} \\
  x &:= 1 \\
  \text{end}
  \end{align*}
  \]

- Assume \( b \) is more sensitive than \( x \)
- With a runtime check
  - \( x=1 \), then \( b \) is obviously leaked, but not if \( x=0 \)
- Need a static analysis to detect
Static and Dynamic Binding

- **Static binding**
  - Security class of an object is fixed
  - This is the case for BLP and Biba
  - This is the case for most system models

- **Dynamic binding**
  - Security class of an object can change
  - For \( b = a \), then the security class of \( b \) is \( b/a \)
  - E.g., High-water mark secrecy, LOMAC, IX, …
Semantics

- Program is secure if:
  - Explicit flow from S is secure
  - Explicit flow of all statements in a sequence are secure (e.g., S1; S2)
  - Conditional $c: S_1, ..., S_m$ is secure if:
    - The explicit flows of all statements $S_1, ..., S_m$ are secure
    - The implicit flows between $c$ and the objects in $S_i$ are secure
Type Safety

- A type-safe language maintains the semantics of types. E.g., can't add int's to Object's.

- Type-safety is compositional. A function promises to maintain type safety.

Example 1
Object obj;
int i;
obj = obj × i;

Example 2
String proc_obj(Object o);
...
main()
{
    Object obj;
    String s = proc_obj(obj);
    ...
}
Security Types

Example 1
int{high} h1,h2;
int{low} l;
l = 5;
h2 = l;
h1 = h2 + 10;
l ≠ h2 + l;

• Key insight:
  label types with security levels

• Security-typing is compositional

Example 2
String{low}
proc(Object{high} o);
...
main()
{
  Object{high} obj;
  String{low} s;
  s = proc_obj(obj);
  ...
}
Decentralized Label Model

- Labels have *owners* and *readers*
  - **Owner**: whose data was observed to generate value
  - **Reader**: principals allowed by an owner to read
  - Readers are specified by each owner

- Label representation
  - \( L = \{o1: r1, r2; o2: r2, r3\} \)

- Channel
  - Values are written to *output channels*
  - Each channel has a set of readers

- Effective Readers
  - Intersection of all reader sets of the label
  - Effective readers of \( L \) are \( \{r2\} \) because only it can read from \( o1 \) and \( o2 \)

- Act for
  - Readers can “act for” others, using their permissions

- Semantics
  - A value can be written to a channel only if each channel reader has authority to act for some effective reader for the value
  - *Permission Semantics*
Figure 1: Medical Study Scenario
Example

Figure 2: Bank Scenario
Relabeling Semantics

• Basics
  ‣ Assignment causes a relabel of value
  ‣ Default is *rest*ric*tion* according to *-property
    • A new label contains the owners of the old, but same or fewer readers

• *Declass*ification* semantics
  ‣ An authority for an owner can
    • Remove that owner
    • Add readers for that owner
Combination Semantics

- **Join** (e.g., multiply 2 numbers)
  - Assign value of label L to variable with value of label L’ results in a join of L and L’
  - Least restrictive combination
  - Least upper bound
  - Union owners and intersect readers

- **Meet** (dual of join):
  - Most restrictive label that can apply to each input for join to be possible
  - Greatest lower bound
  - Fewest readers to achieve join label, most owners…
Label Hierarchies

- Acts-for defines a hierarchy
  - HMO acts-for A
  - B acts-for doctors
  - Secret acts-for classified

- Labels as flows -- Forms an information flow lattice

- Constraints
  - Reader constraint: flows contain \((o, r)\) and \(r'\) acts-for \(r\), then set contains \((o, r')\)
  - Owner constraint: flows contain \((o,r)\) and \(o'\) acts-for \(o\), then set contains \((o', r)\)
    - Or flow set does not contain \((o', r)\) and \(o'\) acts-for \(o\), then set does not contain \((o, r)\)
Example

Hierarchy
E acts for p:
Changes owner to R

Declassify
R removes p
and adds reader S

Figure 1: Medical Study Scenario
Example

Access
C controls its own data

Hierarchy
T acts for C: T removes Ci from owner

Figure 2: Bank Scenario
Language Support

• Java Information Flow (Jif) has runtime and compilers
  ‣ Several applications of Jif have been developed

• Challenge: labeling and error resolution
  ‣ How do you annotate data with security?
  ‣ How do you fix errors?
    • Many occur due to implicit flows

• Research in automatic retrofitting of programs with security type annotations and mediation
Take Away

• Programs may have the authority to protect security-sensitive data
  ‣ OS may allow them to access data with multiple security requirements

• Program data flows for the basis for reasoning about how program authority is used
  ‣ Can secrets flow to public objects? Can untrusted data flow to trusted?

• Denning model defines secure information flow

• DLM model generalizes to arbitrary policies
Sound relabeling

- Based on static hierarchy (actsFor)
- Claim: cannot use static correctness
- Example:
  - L1={docs: pA; B: pA, pB}
  - L2={docs: docs, pA; B: pA, pB}
- If B => docs
  - L2={docs: pA; B: pA, pB} -- B overrules docs
- If pB => docs at runtime
  - L1={docs: pA, pB; B: pA, pB} -- pB is allowed by B
  - Inconsistent
Sound and complete relabeling

• Choices
  ‣ A reader may be dropped from some owner’s reader set
  ‣ A new owner may be added with a reader set
  ‣ A reader may be added when it actsFor an existing reader in reader set
  ‣ An owner may be replaced by an owner that actsFor it

• This is all the sound relabelings

• What does this mean in the previous case?
Meet Semantics Clarified

- Most restrictive label that can be relabeled to both
  - For inference
- Join of all pairwise components
  - Unrelated owners ==> { }
  - Related owners ==> o’ actsFor o
    - \{o: r1, r2\} meet \{o’: r3, r4\} = \{o: r1, r2, r3, r4\}