CIS 610: Advanced Topics in Systems Security
Information Flow Control

Prof. Kevin Butler
Fall 2011
Projects

- Reminder: project updates are due tomorrow
- 3-5 slides describing progress so far
- Recommended: preliminary version of final report
Problem

• A program is trusted to enforce a system’s policy
  ‣ How do we know?
• Integrity models don’t cover this
• So what can we do?
Problem
Problem

Alice - X Window

Remote Client A

Remote Client B
Problem

- Alice - X Window
- Remote Client A
- Remote Client B

Secret
Problem

[Diagram showing a flow between 'Secret' and 'Remote Client A', and another flow from 'Alice - X Window' to 'Remote Client B'].

Oregon Systems Infrastructure Research and Information Security (OSIRIS) Lab
Problem
Problem
Problem
What’s a Program?

- **Program parts**
  - Instructions, Variables, Control Ops, Procedures, Arguments, System calls/Library calls (sources/sinks)

- 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 := getPasswordFromUser();
3. secret_data_2 := getPasswordFromUser();
4. If(secret_data_1 == secret_data_2)
5.   writeToFile(encrypt(secret_data_1));
6. else
7.   writeToOutput(“Passwords do not match”);
Data Flow

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

• The program operations 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
- Remove restrictions
  - Label polymorphism
  - Run-time label checking
Denning’s Lattice Model

- Formalizes information flow models
  - \( \text{FM} = \{N, P, \text{SC}, /, >\} \)
- Shows that the information flow model instances form a lattice
  - \( N \) are objects, \( P \) are processes,
  - \( \{\text{SC}, >\} \) is a partial ordered set,
  - \( \text{SC} \), the set of security classes is finite,
  - \( \text{SC} \) 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 \( c: S_1, \ldots, S_m \)

• Implicit flows only occur in conditionals
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: S1, ..., Sm is secure if:
  - The explicit flows of all statements S1, ..., Sm are secure
  - The implicit flows between c and the objects in Si are secure
Figure 1: Medical Study Scenario
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

```java
Object obj;
int i;
obj = obj + i;
```

Example 2

```java
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;
\[\text{x} \text{h2} + l;\]

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

• Key insight:
  label types with
  security levels

• Security-typing is
  compositional
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
  - \(L = \{o_1: r_1, r_2; o_2: r_2, r_3\}\)

- 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 \(\{r_2\}\) because only it can read from \(o_1\) and \(o_2\)

- 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
Relabeling Semantics

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

• Declassification 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
  - Both sets of owners, union of readers per owner?
  - Requires refinement of unknowns
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

Declassify
R adds
reader S

Figure 1: Medical Study Scenario
Preventing Implicit Flows

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

- 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
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**
Info Flow & the Web

• Web software is buggy
• Attackers find and exploit these bugs
• Data is stolen / Corrupted
  – “USAJobs.gov hit by Monster.com attack, 146,000 people affected”
  – “UN Website is Defaced via SQL Injection”
  – “Payroll Site Closes on Security Worries”
  – “Hacker Accesses Thousands of Personal Data Files at CSU Chico”
  – “FTC Investigates PETCO.com Security Hole”
  – “Major Breach of UCLA’s Computer Files”
  – “Restructured Text Include Directive Does Not Respect ACLs”
• Controlling information flow can help
DIFC Limitations

• Label systems are complex

• Unexpected program behavior
  • Communication can be unreliable (crashing programs, read errs)

• Cannot reuse existing code
  • Drivers, SMP support, standard libraries
Flume

1. Flume: OS Approach to DIFC
   - User-level implementation of DIFC on Linux
   - Simple label system
   - Endpoints: Glue Between Unix API and Labels

2. Application + Evaluation
   - Real Web software secured by Flume
   - Max Krohn’s OKWS server
   - MoinMoin wiki (100K LOC)
open("/hr/LayoffPlans", O_RDONLY);
Three Classes of Processes

**Flume-Oblivious**
- Flume Reference Monitor
- Process $p$
- Linux Kernel

**Unconfined/Mediators**
- Flume Reference Monitor
- Process $p$
- Linux Kernel

**Confined**
- Flume Reference Monitor
- Process $p$
- Linux Kernel
Labels

• Goal: track which secrets a process has seen
• Mechanism: each process gets a secrecy label
  – Label summarizes which categories of data a process is assumed to have seen.
  – Examples:
    • \{“Financial Reports” \}
    • \{“HR Documents” \}
    • \{“Financial Reports” and “HR Documents” \}
Processes & Labels

• Process can create a new tag and can declassify it
• Any process can add tag to label
  ‣ e.g., change_label({Finance}) by process \( p \)
  ‣ change_label({Finance, HR})
  ‣ change_label({Finance}) - declassification by \( p \)
• Can enforce BLP confidentiality rules with labels (no read-down, no write-up)
Endpoints

• Wrappers on communication - IPC happens between these, not between processes

• Endpoints have their own secrecy and integrity labels, assigned to each file descriptor
  ‣ flows can be one-way or bidirectional depending on labels

• Act as input validation mechanisms
Implementation

- 21K LOC (C++) (14K in ref mon)
- LSM for system call interposition in kernel (500 LOC)
- requires modification to libc, linker, client libs
Pluses and Minuses

• What’s good?

• What’s bad?
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