Labels and Information Flow

Robert Soulé

March 21, 2007

Problem Motivation and History

- The military cares about information flow
  - Everyone can read “Unclassified”
  - Few can read “Top Secret”

Problem Motivation and History

- The military cares about information flow
  - Everyone can read "Unclassified"
  - Few can read "Top Secret"
- So computer scientists care about information flow
  - Bell and LaPadula 1973, "No read up, no write down"
  - Denning 1976, Lattice Model
  - The "Orange Book" 1985, Trusted Computer System Evaluation

We Care Too

- We aren't the Dept. of Defense, but it is still important
- How do I know my free software doesn't steal my private data?
  - Passwords
  - Social Security numbers
  - Financial Records

Three Recent Approaches

- JIF - A language based solution
- Asbestos - A kernel based solution
- Hi Star - Also kernel based, re-imagining Asbestos

Java Information Flow (JIF), What Is New and Different?

- The *decentralized label model*.
  - Very different from the DoD's security model
  - Avoids rigid constraints of traditional multilevel security systems
  - Allows users to control their own information flow
  - Declassification is part of the model

JIF - How Does This Work?

- Mutually distrusting users and programs
- All data is labeled
- When you make variable assignments, labels propagate
- Compiler and runtime ensures that information flow rules are not violated

Who Are These Users?

- JIF Principals are owners/readers/writers
- A principal can be a user or a group
- Can be authorized to act for other principals
- "acts for" allows for a delegation of powers
  - declassifies-for
  - reads-for
- These relationships from a principal-hierarchy

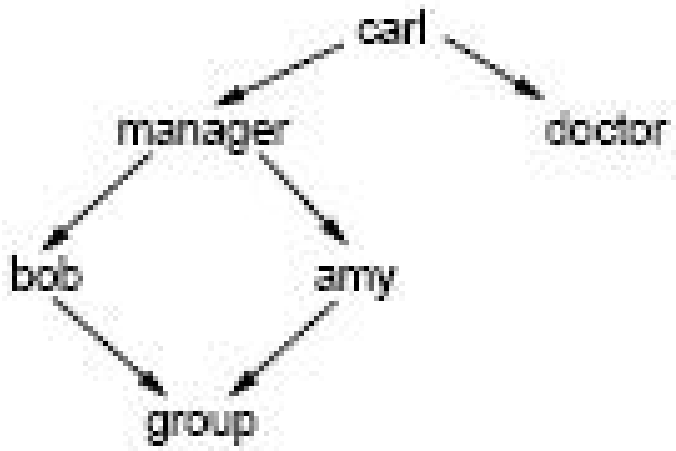

Fig. 3. A principal hierarchy

How Is The Data Labeled?

- Add label constructs to the language
  - Privacy labels restrict read access
  - Integrity labels restrict write access
- Labels and built in Java types form an extended type system
- Checking for type safety enforces flow rules

JIF Labels Details

- Privacy Labels - restrict read access
  - A component (policy) has two parts, owner and readers
    - Owners are the source of the data
    - Readers are possible destinations
  - L = {o1:r1,r2; o2:r2,r3}
- Integrity Labels - restrict write access
  - Components have two parts, owner and writers
    - Owners are the source of the data
    - Writers are possible modifiers
  - L = {o1:w1,w2; o2:w2,w3}

How Do Labels Propagate?

- Every assignment is a relabeling
- Relabeling is permitted according to certain rules
- Let's look at an example:

```
int {Alice:Bob} x; // Alice owns x, Bob can read
int z;
z = x; // z gets x's label

int {Alice:Bob, Chuck} y;
x = y; // OK: policy on x is stronger
y = x; // BAD: policy on y is not as strong as x
```

Relabeling Details

- Privacy labels may be safely changed in four ways:
  - Remove a reader - *more restrictive*
  - Add a policy - *more restrictive*
  - Add a reader - *it is safe to add a reader r' if there is already a reader r, and r' acts for r.*
  - Replace an owner - *it is safe to replace an owner o with a new owner o' if o acts for o.*
- Integrity labels may be safely changed in four ways:
  - Add a writer - *more restricted in subsequent use*
  - Remove a policy - *restricts the number of allowed writers.*
  - Replace a writer -*it is safe to replace a writer w' with a writer w if w' acts for w (really adding a writer).*
  - Add a policy - *only if the new policy J is more restrictive than the old policy I.*

Label Notation Basics

- Introduce a formalism so we can reason about flow
- The notation $L_1 \sqsubseteq L_2$ means L_1 is at most as restrictive as L_2.
- Some policy label examples:
  - $\{amy : bob, carl\} \sqsubseteq \{amy : carl\}$
  - $\{amy : manager\} \not\sqsubseteq \{amy : bob\}$
- Relabeling is permitted only if labels become more restrictive.

So What Is z's Label?

- Recall:

```
int {Alice:Bob} x;
int {Alice:Bob, Chuck} y;
int z = x + y;
```

- The new value is the least upper bound or join

$$L_1 \sqsubseteq L_1 \sqcup L_2$$

- $\{amy : bob\} \sqcup \{amy : bob, chuck\} = \{amy : bob; amy : bob, chuck\} = \{amy : bob\}$

How Do I System.out.println?

- Additional mechanism: the label $\{\}$ is used for raw output channels.
- Data can only be written only if it has no privacy restrictions.
- Example: data labeled $\{bob : bob\}$ cannot be written to the network because $\{bob : bob\} \not\sqsubseteq \{\}$
- We need a way to declassify this data so that it can be written.

Declassification for Privacy Labels

- A process is authorized to act on behalf of some set of principals
- This set is the *authority* of a process
- Data can be declassified with respect to a policy owned by a principal
- L_1 may be declassified to L_2 when

$$L_1 \sqsubseteq L_2 \sqcup L_A$$

where L_A is a label containing exactly the policies of the form $\{p{:}\}$ for every principal p in the current authority

Declassification for Integrity Labels

- L_1 may be declassified to L_2 when

$$L_2 \sqcap L_A^I \sqsubseteq L_1$$

  where L_A^I is an integrity label in which there is a policy for every principal in the authority of the process.
- $\sqcap$ is the meet or greatest lower bound

Jif Language Overview

- Jif extends Java, supports:
  - Mutable objects, Subclasses, Dynamic type tests, Access control, Exceptions
- Every expression has a labeled type
- a `declassify` operator
- An `actsFor` statement
- A `switch label` statement - *if label is x, do this...*
- Procedure call may grant authority possessed by the caller

Implicit Flows

- Use the program counter to control implicit information flows
- An expression is always at least as restrictive as the pc label
- Consider the code:

```
x = 0;
if (b) {
   x = 1;
}
```

- Initially the *pc* is $\{\}$
- At *if(b)* the *pc* is $\{b\}$
- At the assignment, enforce $\{b\} \sqsubseteq \{x\}$

Contributions and Limitations

- Contributions
  - Decentralizes authority
  - Focus on a usable programming model
  - Makes information flow accessible to developers
- Limitations
  - Assumes a trusted execution platform
  - Assumes a trusted compiler.
  - Assumes a principle hierarchy. Cannot express arbitrary, non-hierarchical relationships

Another Approach

- JIF offers a solution at the language level
  - Allows for fine granularity
  - Enforces a choice of language and compiler
- Perhaps this isn't the right place?

Asbestos

- Operating system level solution
- Supports server applications that keeps users isolated
  - A web server is the running example
  - Each application defines its own policies
- Every process has a *send* and *receive* label
- Processed communicate using messages sent to ports
- Kernel ensures that a process is permitted to *send* to another process's *receive* port.

Asbestos Labels

- Labels support decentralized compartments
- A program has discretionary right to declassify data in the compartment it created (similar to capabilities).
- Can give rights to declassify to other trusted programs
- Separate send and receive labels with different defaults

Asbestos Labels

- A label L is a function that maps handles to taint levels
- Handles: 61-bit opaque identifiers
- Levels: { *, 0, 1, 2, 3 }
- Syntax: { handle-1 level-1, handle-2 level-2, ..., default-level }
- Default-level applies to all handles not explicitly mentioned
- For send, * is the most privileged, 1 is the default. For receive, 2 is the default.

Asbestos Label Basics

- Process P can send to Q if

$$P_s \sqsubseteq Q_r$$

- When a message is delivered to Q, Q's send label is contaminated by P,

$$Q_s \leftarrow Q_s \sqcup P_s$$

Discretionary Contamination (Optional Detail)

- A process may want to make a message more tainted C_2
- This is okay, it doesn't violate flow properties
- The sender's new label becomes

$$E_s = P_s \sqcup C_s$$

- Q's send label is contaminated by E_s

$$E_s \sqsubseteq Q_r$$

$$Q_s \leftarrow Q_s \sqcup E_s$$

Declassification

- A process with $P_s(h) = *$ has declassification privilege for h
- If P receives a message from process Q with $Q_s(h) = 3$, $P_s(h)$ remains $*$.
- P can forward data from Q with less taint

$$Q_s \leftarrow Q_s \sqcup (E_s \sqcap Q_s*)$$

- $(E_s \sqcap Q_s*)$ has precedence

Decontamination (Optional Detail)

- A process with privilege can distribute privilege to other processes
  - Note that this is different from Jif, which has a fixed hierarchy of users controlling I/O channels
- A process can decontaminate another process by lowering their send label and raising their receive label
- This makes the system more permissive
- P can forward data from Q with less taint

$$E_s \sqsubseteq Q_s \sqcup D_r$$

$$Q_s \leftarrow (Q_s \sqcap D_s) \sqcup (E_s \sqcap Q_s*), Q_r \leftarrow Q_r \sqcup D_r$$

- Where D_s is the decontamination send label and D_r is the decontamination receive label

Integrity (Optional Detail)

- A process may speak for a user u and therefore write to u's file
- This is a positive right, not a taint
- Asbestos must verify that a process speaks for u before permitting
- A verification label temporarily restricts the receiver's effective receive label

$$E_s \sqsubseteq (Q_r \sqcup D_r) \sqcap V$$

- where V is the verification label

Event Processes Continued

- Processes quickly become over contaminated
- *Event processes* abstracts the notion of a single user's process state
- Associated with a base process
- Kernel state consists of:
  - send label,
  - receive label
  - port receive rights,
  - private set of memory pages

OK Web Server

- Demultiplexor process accepts connection, and hands it off
- One of several worker processes service the request
- OKWS isolates services in different processes
- Asbestos provides additional isolation within the process with event processes
- Supports database access through a proxy
  - Adds a "user ID" column to underlying database

Limitations

- Label operations must be performed with every IPC (is this slow?)
- How are labels propagated between nodes?
- Limits choice of concurrency
- Size of the labels in the system increase with the number of sessions/users

Hi Star - What Was Wrong With Asbestos?

- Different goals: Unix vs. specialized web server
- Hi Star closes covert channels inherent to Asbestos
  - mutable labels, IPC
- Lower Level kernel interface
  - Process vs. Container+Thread+AS+Segments+Gates
  - One third the kernel code
  - Adds generality with user-space Unix Library
- System-wide support for persistent storage
  - Asbestos uses trusted user-space file server
- Resources are manageable
  - In Asbestos, had to reboot to kill a runaway process

* *source: OSDI 2006 Hi Star slides*

Labels in Hi Star

- Hi Star uses Asbestos' labeling system, with one clarification
- Recall the Asbestos Label levels:
  - * has untainting privileges
  - 0 cannot be written/modified by default
  - 1 default level , no restriction
  - 2 cannot be untainted/ exported by default
  - 3 cannot be read/observed by default

Labels in Hi Star Continued

- * denoted untainting privilege, not a taint (breaks lattice)
- Acts in two contexts
  - When T reads an object, treated as higher than a number
  - When T writes an object, treated as less than a number
- To avoid confusion, use a new high star symbol, ✪,
- Only appears in the notation, not in the labels
  - $* < 1 < 2 < 3 <$ ✪

Labels: New Rules

- T observes O requires:

$$L_O \sqsubseteq L_T^{\star}$$

- T modifies O requires:

$$L_T \sqsubseteq L_O \sqsubseteq L_T^{\star}$$

- Note this affects how much T must raise label to observe object O
  - Before said lowest possible value was $L_O \sqcup L_P$
  - Now lowest value is $(L_O \sqcup L_T^{\star})*$
  - i.e., "T can keep its stars"

Kernel Design

- Six object types
  - Segment (Data), Threads, Containers (Directory), Address Space, Gate (IPC), Device (Network)
- Each object has:
  - 61 bit object ID
  - Label
  - Quota, to bound storage usage
  - 64 bytes of metadata (modification time, etc.)
  - Flags
    - Immutable flag makes the object read only

Single Level Store

- Single-level means no distinction between memory and secondary storage
- View memory as just a cache for disk–everything persists across reboots
- Solves the system initialization problem without a superuser
- Otherwise, how would users get their *s back after a reboot?

No Superuser

- Root is a big security hole for information flow
  - Can read/modify anything, violating any policy
- But then can you create a process you can't kill?
  - No–because allocation uses container hierarchy
  - Any object you create is "charged" against a container's space quota
- Idea: no implicit resource allocation

Resource Exhaustion

- This is troublesome for both Asbestos and Hi Star
- The single level store and quotas help ameliorate this problem
- Quotas form a hierarchy under control of the system administrator
- This is, notably, the only inherent hierarchy in Hi Star

The wrap Program

- 110 line trusted program
- Has untainting privileges
  - Example: untaint the virus scanner's results, and report back to the user
- A program cannot read tainted data unless first tainting itself
- If wrap is correctly implemented, program launched by wrap cannot leak information

Emulating UNIX Environment

- Implemented as a library in user space
  - Information such as the exit status of the child process is made explicit by the UNIX library
- Hi Star file system uses segments and containers
- Files map to segments, directories to containers
- Processes are user space conventions
  - Actually map to containers, segments, gates, threads, and address space objects
- File descriptors are mapped to segments

Emulating UNIX Permissions

- World-readable file, only writable by owner?
  - Each user u owns two categories, u_r and u_w, gets * on login
  - Set file's label to $\{u_w 0, 1\}$
  - Now default process with label $\{1\}$ can't write file, but can read
- Group-readable file, only writable by owner?
  - Have to introduce categories g_r, g_w for group,
  - give $g_r *, g_w *$ on login
  - Set file label to $\{g_r 3, u_w 0, 1\}$

IPC

- Gates provide the mechanism for IPC
- Example: Timestamped digital signature daemon (pg. 271)
  - Daemon knows secret signature key labeled $\{d_r 3, d_w 0, 1\}$
  - Daemon creates a service gate G with $\{d_r *, d_w *, 1\}$
  - Client running with process categories $p_r *, p_w *$
  - Client allocates a new category r
  - Client allocates a "return gate" G_r with label $\{p_r *, p_w *, 1\}$, clearance $\{r0, 2\}$
  - Client jumps through gate G, starts server code with $d_r *, d_w *$
  - Server code now has access to secret key, computes signature
  - Returns to client by jumping through G_r; resets thread's labe from $\{d_r *, d_w *, r *, 1\} \rightarrow \{p_r *, d_w *, r *, 1\}$

Signals

- Implemented by sending an alert to a thread in a process
- Requires the permission to modify the thread's address space object
- Each process exposes a signal gate
- The gate has label $\{p_r*, p_w*, 1\}$
- The clearance of the gate is $\{u_w 0, 2\}$
- Only threads that have the user's privilege can send signals to that user's process

Authentication

- Hi Star authenticates without any highly trusted process
- Users may supply their own authentication service
- Four separate entities coordinate to authenticate a user
- Login client, directory service, per-user authentication service, logging service