Information Flow Control for Standard OS Abstractions

Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris
Problem Motivation and History

- Applications are very complex
  - Increasingly rely on third-party software
  - Can’t audit all the code
- How do I know my software doesn’t steal my private data?
  - Social Security numbers
  - Financial Records
  - Medical Records
A Well-Established Approach

- Minimize the trusted computing base
- Use labels to track information flow
  - Bell and LaPadula 1973, “No read up, no write down”
  - Biba 1976, Integrity Model
  - Denning 1976, Lattice Model
  - The “Orange Book” 1985, Trusted Computer System Evaluation
  - Meyers and Liskov 2000, Decentralized label model
What is 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
The Usual Suspects

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

- And some objections
  - Do I want to re-write all my applications?
  - I don’t want to switch to a new operating system.
Can we ensure the same privacy/integrity guarantees for existing operating systems and languages?

Flume says "yes", but with some assumptions
- No bugs that result in super user access
- Processes do not leak data through timing attacks
The Flume model gives general guidelines about what properties the system must uphold to be considered “secure.”

The Flume system are the system specifics such as what API two processes use to communicate.
The Flume Model: Tags and Labels

- \( \mathcal{T} \) is a set of opaque tokens called tags
- A tag carries no inherent meaning, but are usually associated with some category of secrecy or integrity
  - Tag \( b \) might be Bob’s private data
Labels are subsets of $\mathcal{I}$

Each flume process $p$ has two labels:
  - $S_p$ for secrecy
  - $I_p$ for integrity

If $t \in S_p$ then the system assumes $p$ has some private data

If $t \in I_p$ then every input to $p$ has been endorsed as having integrity $t$
The Flume Model: Secrecy

- If tag $b$ represents Bob’s secret data
- Flume ensures four guarantees:
  - If process $p$ reads the secret data, then $b \in S_p$
  - $p$ with $b \in S_p$ can only write to process or file $q$ if $b \in S_q$
  - $p$ cannot remove $b$ from $S_b$
  - $p$ with $b \in S_p$ cannot transmit information over an uncontrolled channel (like the network)
The Flume Model: Integrity

- If tag $v$ represents data that is vendor certified
- Flume ensures four guarantees:
  - If process $p$ modifies data, then $v \in I_p$
  - A process $p$ with $v \in I_p$ cannot read from files or processes that lack $v$ integrity
  - A process $p$ cannot add $v$ to $I_p$
  - $p$ with $v \in I_p$ cannot accept input from an uncontrolled channel (like the network)
The Flume Model: Decentralized Privilege

- Every process $p$ owns a set of capabilities $O_p$
- Each tag $t$ has two associated capabilities:
  - $t^+$: If $t^+ \in O_p$ then $p$ can add $t$ to its labels
  - $t^-$: If $t^- \in O_p$ then $p$ can remove $t$ from its labels
- In terms of secrecy
  - $t^+$ lets a process grant itself privilege to receive secret $t$ data
  - $t^-$ lets a process remove $t$, declassifying the data
- In terms of integrity
  - $t^+$ lets a process add $t$ to some data, endorsing the data as high-$t$-integrity
  - $t^-$ lets a process remove $t$, allowing it to receive low-$t$-integrity data
- Note that any process can allocate a tag, and it is granted $t^+$ and $t^-$
The Flume Model: Global Capabilities

- Flume also supports the global capability set $O$
- All processes own every capability in $O$
- A process can test if a capability is in $O$, but cannot enumerate all capabilities in $O$
Security

*Definition 1.* A system is secure in the Flume model if and only if all allowed process label changes are “safe” (Definition 2) and all allowed messages are “safe” (Definition 3).
Only process $p$ can change $S_p$ and $I_p$ (Like Hi Star, different from Asbestos). Can change labels if:

**Definition 2.** For a process $p$, let $L$ be $S_p$ or $I_p$, and let $L'$ be the new value of the label. The change from $L$ to $L'$ is safe if and only if:

\[
\{L' - L\}^+ \cup \{L' - L\}^- \subseteq O_p
\]
Security: Safe Messages

Flume restricts messages in the classical manner ("no read up, no write down"). If a process has \( t^+ \) and \( t^- \) it has dual privilege \( D_p \). If two processes could change their labels to communicate, then communication is permitted, according to the definition below:

**Definition 3.** A message from \( p \) to \( q \) is safe if and only if

\[
S_p - D_p \subseteq S_q \cup D_q
\]

and

\[
I_q - D_q \subseteq I_p \cup D_p
\]
Terminals, printers, etc. have labels: $S_x = I_x = \{\}$

Files and directories are modeled as processes with immutable secrecy and integrity labels
Flume System: Endpoints

- Flume applies DIFC controls to the Unix communication primitive, the file descriptor.
- Flume assigns an endpoint to each Unix file descriptor.
- When a process $p$ acquires a new file descriptor, it gets the corresponding endpoint $e$.
- $e$ has $S_e = S_p$ and $I_e = I_p$. 
Definition 4. A readable endpoint $e$ is safe iff

$$(S_e - S_p) \cup (I_p - I_e) \subseteq D_p$$

A writeable endpoint is safe iff

$$(S_p - S_e) \cup (I_e - I_p) \subseteq D_p$$
Flume System: Endpoints

All communication occurs between two endpoints. So, Definition 3 becomes:

*Definition 5.* A message from endpoint $e$ to endpoint $f$ is safe iff $e$ is writeable, $f$ is readable, $S_e \subseteq S_f$, and $I_f \subseteq I_e$. 
Flume System: Enforcing Safe Communication

For communication that Flume can completely control (IPC):

- The Flume reference monitor proxies a pipe or a socket between two Flume processes.
- When \( p \) send to \( q \), Flume checks the corresponding endpoint labels.
- The reference monitor silently drops data that is unsafe.
- \( p \) and \( q \) can change the endpoint labels so long as the changes are safe.
For communication that Flume cannot completely control (File I/O):

- Once Flume allows a process to open a file for reading or writing, it allows all future reads or writes
- The reference monitor performs open on behalf of the process, and returns the file descriptor and an immutable endpoint
Implementation

- Implemented in user-space
- Linux version has a small component in the kernel for system call interposition
- Dedicated spawner process
- Remote tag registry (for persistence for capabilities)
- User space file servers
- Flume aware C library redirects Unix system calls to the reference manager
Confined vs. Unconfined

- Processes in Flume are either confined or unconfined
- Unconfined processes have empty labels and empty non-global ownership
- The reference monitor assigns them this immutable endpoint
- An unconfined process may circumvent the Flume RM (Linux dictates the security)
Confined vs. Unconfined

- Confined processes are those controlled through the RM
- Flume’s spawn program creates new confined processes (combines fork and exec)
- Confined processes may not fork
- Flume offers flume_pipe() and flume_socketpair() for sharing pipes and sockets
- Confined processes are disallowed direct access to the file system
Files and Directories

- Files are opened with Flume’s open, regular close
- Directories are assigned a label, enforces reading directory’s contents
- Typically, directories become more secret as one moves away from the root
- Integrity is non-increasing down a path (so the path has the same integrity as the file)
- Special /ihome directory cannot directly open or read, but can traverse
Persistence

- A tag repository maintains 3 persistent databases
  - maps login to capabilities
  - capability groups
  - file attributes
Sample Application

- Implemented a Flume version of MoinMoin wiki
- Moves all the trusted code to one small implementation
Security Evaluation

- Three ACL bugs in the MoinMoin implementation they forked
- One was disabled, the other two were prevented
Flume Overhead

- FlumeWiki is 43 percent slower than MoinMoin in read throughput
- FlumeWiki is 34 percent slower than MoinMoin in write throughput
- Adds a latency overhead of 40 ms