Information Flow Control For Standard OS Abstractions

Maxwell Krohn, Alex Yip, Micah Brodsky, Natan Cliffer, Frans Kaashoek, Eddie Kohler, Robert Morris

MIT
SOSP 2007

Presenter: Lei Xia
Mar. 2 2009
Outline

- Motivation
- Flume Model
- Flume System
- Evaluation
- Conclusion
Motivation

- Protecting confidential data in computing environments
- Access controls are insufficient to regulate the propagation of information after it has been released for processing by a program
Information Flow Security

- Track and regulate the information flows of the system
  - Prevent secret data from leaking to unauthorized processes (secrecy)
  - Prevent untrusted software to be compromised through malicious inputs (integrity)
Decentralized Information Flow Control (DIFC)

- Share information with untrusted code
- Control how untrusted code disseminates the shared information to others
- Support for declassification of information in a decentralized way
- Improves the security of complex applications even in the presence of potential exploits
Previous Work

- Programming language abstractions
  - Jif: provide more fine-grained control at the granularity of functions in processes
  - But requires applications to be rewritten
- Integrated into communication primitives in OS kernel
  - Asbestos and HiStar Operating Systems
  - Granularity of unreliable messages between processes (Asbestos) or segments (HiStar)
Flume

- Implements a user-level reference monitor
- Provides DIFC at the granularity of processes
- Integrates DIFC controls with standard communication channels like pipe, sockets, file descriptors
- Simple label system
Flume Model - Tags and Labels

- Flume model closely follow IFC, add new representation
- Each tag is associated with some category of secrecy or integrity
- Labels are subsets of Tags
  - Form a lattice under partial order of subset relation
- Used for tracking
Flume Model - Secrecy and Integrity

- Each Flume process $p$ has two labels
  - $S_p$ for secrecy
  - $I_p$ for integrity
- If tag $t \in S_p$, then it is assumed that process $p$ has seen private data tagged with $t$
- If $t \in I_p$, then every input to $p$ has been endorsed as having integrity for $t$
- Files and objects also have secrecy and integrity labels
Decentralized Privilege

- Any process can create new tags
  - Gets the privilege to declassify and/or endorse information for those tags
- Two capabilities per tag
  - $t+$: Ability to add $t$ to the label
  - $t-$: Ability to remove $t$ from the label
- Each process $p$ owns a set of capabilities $O_p$
Capabilities

- Dual privilege
  \[ D_p = \{ t \mid t^+ \in O_p \text{ and } t^- \in O_p \} \]

- A process \( p \) allocating a new tag \( t \)
  \[ O_p = O_p \cup \{ t^+, t^- \} \]

- Global capability set \( O \)
  \[ \text{System enforces that } O \subseteq O_p \text{ for all } p \]
  \[ \text{Only tag allocation can change } O \]
Flume Model: Security – Safe Messages

- Restriction of process communication to prevent data leaks

- $p$ can send a message to $q$ only if,
  - $S_p \subseteq S_q$ (less secret to more secret -> allow)
  - $I_q \subseteq I_p$ (more integrity to less integrity -> allow)

- A message from $p$ to $q$ is safe iff,
  - $S_p - D_p \subseteq S_q \cup D_q$ and
  - $I_q - D_q \subseteq I_p \cup D_p$
Secrecy and Integrity Protections

- Export protection
  - Secrecy tag $t^+$ is added to $O$
  - Only process with $t^-$ can 'declassify'

- Read protection
  - Controlling $t^+$, thereby limiting the processes

- Integrity protection
  - integrity tag $\nu^-$ is added to $O$
  - Only a certifier who has $\nu^+$ can 'endorse'
Secret data with tag ‘b’

Read

Process P

b ∈ Sp

Write

Any process can read secret b since b+ is in O

Process Q

Process R

If b ∈ Sp, process p cannot declassify this data until it has b- ∈ Op

Sr = {}
Integrity - Illustrated

Integrity Protection

\( v \in I_P \) (Certifier approved)

- **Write** Certifier Approved
- **Read** (Inputs to process P)

- Process P
- Process Q
- Process R

- \( v \in I_Q \)
- \( I_R = \{ \} \)

Process R cannot add \( v \) to \( I_R \) since \( v \) is not in \( O \)

Only certifier has \( v+ \) to endorse data
Security – Safe Label Changes

- External Sinks and Sources
  - Remote host, terminal, sockets…
  - $S_x = I_x = {}$

- Objects
  - Assigned immutable secrecy and integrity labels
  - Creating process specifies these labels
Security – Safe Label Changes

- In Flume, only process $p$ can change $S_p$ or $I_p$ and must request such a change explicitly.
- For a process $p$, $L$ be $S_p$ or $I_p$, $L'$ be the new label.
- Change from $L$ to $L'$ is safe iff,

$$
\forall \{L' - L\}^+ \cup \{L - L'\}^- \subseteq Op
$$
Flume System: Endpoint abstraction

- Need to apply DIFC controls to existing APIs
- Glue between flume and standard communication abstractions like sockets, file descriptors
- Flume assigns an endpoint to each Unix file descriptor
- A process can potentially adjust the labels on an endpoint
- All IPC happens between two endpoints
Flume System: Endpoints

- A process owns readable/writable/both endpoints for each of its resource
- A readable endpoint is safe iff
  \[ (S_e - S_p) \cup (I_p - I_e) \subseteq D_p \]
- A writable endpoint is safe iff
  \[ (S_p - S_e) \cup (I_e - I_p) \subseteq D_p \]
- Safe flow between endpoints ensures safe flow between processes
Examples – IPC communication

Figure 1: Processes $p$ and $q$. Assume $O = \emptyset$. 

- $S_p = \{x, y\}$
- $O_p = \{y^+, y^-, z^+\}$

- $S_q = \emptyset$
- $O_q = \{x^+, x^-, y^+\}$
Examples – Shell and Editor

Figure 2: A configuration for Bob’s shell and editor. Here, $O = \{b^+\}$. 
Flume Implementation

- Linux Security Model implements Flume’s system call interposition
- Reference Monitor keeps track of each process’s labels, authorizes or denies its requests to change labels and handles system calls on its behalf
- RM has the following components
  - Spawner process
  - Remote tag registry
  - User space file servers
- Flume aware libc does system call interposition
Figure 3: High-level design of the Flume implementation. The shaded boxes represent Flume’s trusted computing base.
Spawner process

- The reference monitor calls spawner which calls fork
- In the child process, the spawner
  - Enables the Flume LSM policy
  - Performs any setlabel label manipulations if the file to execute is setlabel
  - Opens the requested executable (e.g. foo.sh), interpreter (e.g. /bin/sh) and dynamic linker (e.g., /lib/ld.so) via standard Flume open calls, invoking all of Flume’s permission checks;
  - Closes all open file descriptors except for its control socket and those opened in the previous step
  - Claims any file descriptors by token
  - Calls exec
Limitations

- Bigger TCB
  - Linux stack (Kernel + glibc + linker)
  - Reference monitor (~21 kLOC)
- Confined processes like MoinMoin don’t get full POSIX API.
  - `spawn()` instead of `fork()` & `exec()`
  - `flume_pipe()` instead of `pipe()`
Case Study – Moin Moin Wiki

- Python based web publishing system
- Designed to share documents
- Each page can have an ACL
- 91 K LOC!

Figure 7: Label setup for a read or write request in FlumeWiki. wiki.py only gets capability $w_u^+$ if writing. The target page is export- and write-protected by user $u$. 
Case Study – Overhead

- 1000 LOC launcher/declassifier
- 1000 out of 100K LOC in MoinMoin changed
- Python interpreter, Apache unchanged
- Two ACL bugs are not exploitable in Flume
- Performs within a factor of 2 of the original on read and write tests
- Latency and throughput within 45% and 35% of the unmodified MoinMoin wiki, respectively
Case Study – Interposition Overhead

- For most system calls, Flume adds 35–286ms per system call which results in latency overhead of a factor of 4–35
- Additional 2 system calls account for approximately 40ms of Flume’s additional latency
- An IPC round trip takes 12 system calls on Flume, incurring the three-fold performance penalty for additional system calls
## Performance – System calls

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linux</th>
<th>Flume</th>
<th>diff.</th>
<th>mult.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkdir</td>
<td>86.0</td>
<td>371.1</td>
<td>285.2</td>
<td>4.3</td>
</tr>
<tr>
<td>rmdir</td>
<td>13.8</td>
<td>106.8</td>
<td>93.0</td>
<td>7.7</td>
</tr>
<tr>
<td>open</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>— create</td>
<td>12.5</td>
<td>200.2</td>
<td>187.7</td>
<td>16.0</td>
</tr>
<tr>
<td>— exists</td>
<td>3.2</td>
<td>110.3</td>
<td>107.1</td>
<td>34.5</td>
</tr>
<tr>
<td>— exists, inlined</td>
<td>3.3</td>
<td>41.0</td>
<td>37.7</td>
<td>12.5</td>
</tr>
<tr>
<td>— does not exist</td>
<td>4.3</td>
<td>101.4</td>
<td>97.1</td>
<td>23.6</td>
</tr>
<tr>
<td>— does not exist, inlined</td>
<td>4.2</td>
<td>39.8</td>
<td>35.6</td>
<td>9.5</td>
</tr>
<tr>
<td>stat</td>
<td>2.8</td>
<td>98.1</td>
<td>95.3</td>
<td>34.5</td>
</tr>
<tr>
<td>— inlined</td>
<td>2.8</td>
<td>38.7</td>
<td>35.9</td>
<td>13.7</td>
</tr>
<tr>
<td>close</td>
<td>0.6</td>
<td>0.9</td>
<td>0.2</td>
<td>1.3</td>
</tr>
<tr>
<td>unlink</td>
<td>15.4</td>
<td>110.0</td>
<td>94.6</td>
<td>7.2</td>
</tr>
<tr>
<td>symlink</td>
<td>9.5</td>
<td>106.8</td>
<td>97.3</td>
<td>11.2</td>
</tr>
<tr>
<td>readlink</td>
<td>2.7</td>
<td>90.2</td>
<td>87.5</td>
<td>33.0</td>
</tr>
<tr>
<td>create_tag</td>
<td></td>
<td>22.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>change_label</td>
<td></td>
<td>55.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flumenull</td>
<td></td>
<td>20.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPC round trip latency</td>
<td>4.1</td>
<td>33.8</td>
<td>29.8</td>
<td>8.2</td>
</tr>
<tr>
<td>IPC bandwidth</td>
<td>2945</td>
<td>937</td>
<td>2008</td>
<td>3.1</td>
</tr>
</tbody>
</table>
Performance – FlumeWiki

<table>
<thead>
<tr>
<th></th>
<th>Throughput (req/sec)</th>
<th>Latency (ms/req)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MoinMoin</td>
<td>FlumeWiki</td>
</tr>
<tr>
<td>Read</td>
<td>33.2</td>
<td>18.8</td>
</tr>
<tr>
<td>Write</td>
<td>16.9</td>
<td>11.1</td>
</tr>
</tbody>
</table>

**Figure 9**: Latency and throughput for FlumeWiki and unmodified MoinMoin averaged over 10,000 requests.
Results

Does Flume allow adoption of Unix software?
- 1,000 LOC launcher/declassifier
- 1,000 out of 100,000 LOC in MoinMoin changed
- Python interpreter, Apache, unchanged

Does Flume solve security vulnerabilities?
- Without our knowing, we inherited two ACL bypass bugs from MoinMoin
- Both are not exploitable in Flume’s MoinMoin

Does Flume perform reasonably?
- Performs within a factor of 2 of the original on read and write benchmarks
Conclusion

- DIFC is a challenge to Programmers
- Flume: DIFC in User-Level
  - Preserves legacy software
  - Complements today’s programming techniques
- MoinMoin Wiki: Flume works as promised
Thank you!