Retrofitting Legacy Code for Authorization Policy Enforcement

Vinod Ganapathy
Ph.D. Thesis Defense
Thursday, July 12th, 2007
Principle of Design for Security

To create a secure system, design it to be secure from the ground up

- Historic example:
  - MULTICS [Corbato et al. '65]

- More recent examples:
  - Operating systems
  - Database servers
Relevance of the Principle today

Most deployed software is not designed for security

- Deadline-driven software development
  - Design.Build.(Patch)* is here to stay
- Diverse/Evolving security requirements
  - MULTICS security study [Karger and Schell, ‘72]
Retrofitting legacy code

Need systematic techniques to retrofit legacy code for security

Legacy code → Retrofitted code

INSECURE → SECURE
Retrofitting legacy code

Need systematic techniques to retrofit legacy code for security

- Enforcing type safety
  - CCured [Necula et al., ’02]

- Partitioning for privilege separation
  - PrivTrans [Brumley and Song, ’04]

- Enforcing authorization policies
Enforcing authorization policies

Resource manager

Reference monitor

Allowed?

YES/NO

Resource user

Operation request

Response

⟨Alice, /etc/passwd, File_Read⟩
Retrofitting for authorization

- Mandatory access control for Linux
  - Linux Security Modules [Wright et al.,’02]
  - SELinux [Loscocco and Smalley,’01]

- **Painstaking, manual procedure**
  - Trusted X, Compartmented-mode workstation, X11/SELinux [Epstein et al.,’90][Berger et al.,’90][Kilpatrick et al.,’03]

- Java Virtual Machine/SELinux [Fletcher,‘06]
- IBM Websphere/SELinux [Hocking et al.,’06]
Thesis statement

Program analysis and transformation techniques offer a principled and automated way to retrofit legacy code with reference monitors.
Contributions

Analyses and transformations for authorization policy enforcement

- **Fingerprints**: A new representation for security-sensitive operations
- **Two algorithms** to mine fingerprints
- **Result**: Reduced effort to retrofit legacy code for authorization policy enforcement
  - Manual effort needed reduces to a few hours
  - Applied to X server, Linux kernel, PennMUSH
Outline

- Motivation
- Problem
  - Example
  - Retrofitting legacy code: Lifecycle
- Solution
X server with multiple X clients

Welcome to ABC Bank

Account #: alice123

Password: *************
Malicious remote X client

Welcome to ABC Bank

Account #: alice123

Password: ************
Undesirable information flow

Welcome to ABC Bank

Account #: alice123
Password: ************
Desirable information flow
Other policies to enforce

- Prevent unauthorized
  - Copy and paste
  - Modification of inputs meant for other clients
  - Changes to window settings of other clients
  - Retrieval of bitmaps: Screenshots

[Berger et al., ’90]
[Epstein et al., ‘90]
[Kilpatrick et al., ‘03]
X server with authorization

X client

Operation request

X server

Reference monitor

Allowed?

Authorization policy

Response

YES/NO
Outline

- Motivation
- Problem
  - Example
    - Retrofitting legacy code: Lifecycle
- Solution
Retrofitting lifecycle

1. Identify security-sensitive operations
2. Locate where they are performed in code
3. Instrument these locations

Security-sensitive operations
- Input_Event
- Create
- Destroy
- Copy
- Paste
- Map

Source Code

Policy checks
Can the client receive this Input_Event?
Problems

Manual

- X11/SELinux ~ 2 years [Kilpatrick et al., '03]
- Linux Security Modules ~ 2 years [Wright et al., '02]

Ad hoc

- Violation of complete mediation
- Time-of-check to Time-of-use bugs [Zhang et al., '02] [Jaeger et al., '04]
Our approach

**Principled**

- **Fingerprints**: A new representation of security-sensitive operations

**Automated**

- Legacy code retrofitted using fingerprints
  - Use of static and dynamic program analysis
Approach overview

Legacy code

Miner

Fingerprints

Matcher

Retrofitted code
Outline

- Motivation
- Problem
- Solution
  - Fingerprints
  - Dynamic fingerprint mining
  - Static fingerprint mining [CCS’05]
What are fingerprints?

Code-level signatures of security-sensitive operations

- Resource accesses that are unique to a security-sensitive operation
- Denote key steps needed to perform the security-sensitive operation on a resource
Examples of fingerprints

- \texttt{Input\_Event} :-
  \begin{verbatim}
  Cmp xEvent->type == Keypress
  \end{verbatim}

Security-sensitive operations

\texttt{Input\_Event}
- Create
- Destroy
- Copy
- Paste
- Map

Source Code
Examples of fingerprints

- **Input_Event** :-
  \[ Cmp \ xEvent->type == \texttt{KeyPress} \]

- **Input_Event** :-
  \[ Cmp \ xEvent->type == \texttt{MouseMove} \]

- **Map** :-
  \[ Set \ Window->mapped \ to \ True \ & \]
  \[ Set \ xEvent->type \ to \ \texttt{MapNotify} \]

- **Enumerate** :-
  \[ Read \ Window->firstChild \ & \]
  \[ Read \ Window->nextSib \ & \]
  \[ Cmp \ \text{Window} \neq 0 \]
MapSubWindows(Window *pParent, Client *pClient) {
    Window *pWin;
    ...
    // Run through linked list of child windows
    pWin = pParent->firstChild; ...
    for (;pWin != 0; pWin=pWin->nextSib) {
        ...
        // Code that maps each child window
        ...
    }
}

Performs **Enumerate**

**Enumerate** :- Read Window->firstChild & Read Window->nextSib & Cmp Window ≠ 0

Fingerprint matching
Placing authorization checks

- X server function **MapSubWindows**

```c
MapSubWindows(Window *pParent, Client *pClient) {
    Window *pWin;
    ...
    // Run through linked list of child windows
    if CHECK(pClient, pParent, Enumerate) == ALLOWED {
        pWin = pParent->firstChild; ...
        for (; pWin != 0; pWin = pWin->nextSib) {
            ...
            // Code that maps each child window
            ...
        }
    } else { HANDLE_FAILURE }
}
```
Fingerprint matching

- Currently employ simple pattern matching
- More sophisticated matching possible
  - Metacompilation [Engler et al., ‘01]
  - MOPS [Chen and Wagner, ‘02]
- Inserting authorization checks is akin to static aspect-weaving [Kiczales et al., ’97]
- Other aspect-weaving techniques possible
  - Runtime aspect-weaving
Outline

- Motivation
- Problem
- Solution
  - Fingerprints
  - Dynamic fingerprint mining [Oakland’06]
  - Static fingerprint mining
Dynamic fingerprint mining

Security-sensitive operations

Input_Event
Create
Destroy
Copy
Paste
Map

Source Code

Output: Fingerprints

Input_Event :-

Cmp xEvent->type == KeyPress
Dynamic fingerprint mining

- **Security-sensitive operations**  
  
  | **Input_Event** | Input to window from device |
  | **Create**      | Create new window           |
  | **Destroy**     | Destroy existing window     |
  | **Map**         | Map window to console       |

- Use this information to induce the program to perform security-sensitive operations
Problem definition

- **S**: Set of security-sensitive operations
- **D**: Descriptions of operations in **S**
- **R**: Set of resource accesses
  - *Read/Set/Cmp* of *Window/xEvent*
- Each \( s \in S \) has a fingerprint
  - A fingerprint is a subset of **R**
  - Contains a resource access unique to \( s \)
- **Problem**: Find fingerprints for each security-sensitive operation in **S** using **D**
Traces contain fingerprints

Security-sensitive operations

- **Input_Event**
- Create
- Destroy
- Copy
- Paste
- Map

Source Code

Runtime trace

- Induce security-sensitive operation
  - Typing to window will induce **Input_Event**
- Fingerprint **must** be in runtime trace
  - `Cmp xEvent->type == KeyPress`
Compare traces to localize

Security-sensitive operations

- Input Event
- Create
- Destroy
- Copy
- Paste
- Map

Source Code

Runtime trace

- Localize fingerprint in trace
  - Trace difference and intersection
Runtime traces

- Trace the program and record reads/writes to resource data structures
  - Window and xEvent in our experiments
- Example: from X server startup
  (In function SetWindowToDefaults)

  - Set Window->prevSib to 0
  - Set Window->firstChild to 0
  - Set Window->lastChild to 0

  ...  

  about 1400 such resource accesses
Using traces for fingerprinting

- Obtain traces for each security-sensitive operation
  - Series of controlled tracing experiments

- Examples
  - Typing to keyboard generates \textit{Input\_Event}
  - Creating new window generates \textit{Create}
  - Creating window also generates \textit{Map}
  - Closing existing window generates \textit{Destroy}
### Comparison with “diff” and “∩”

**Annotation is a manual step**

<table>
<thead>
<tr>
<th></th>
<th>Open <code>xterm</code></th>
<th>Close <code>xterm</code></th>
<th>Move <code>xterm</code></th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Unmap</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Input Event</strong></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Comparison with “diff” and “∩”

Perform same set operations on resource accesses

<table>
<thead>
<tr>
<th></th>
<th>Open xterm</th>
<th>Close xterm</th>
<th>Move xterm</th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Unmap</strong></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Input_Event</strong></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Create = Open xterm ∩ Open browser - Move xterm
Set equations

- Each trace has a set of labels
  - Open `xterm`: \{Create, Map\}
  - Browser: \{Create, Destroy, Map, Unmap\}
  - Move `xterm`: \{Map, Input_Event\}

- Need set equation for \{Create\}
  - Compute an exact cover for this set
  - Open `xterm` \( \cap \) Open browser – Move `xterm`

- Perform the same set operations on the set of resource accesses in each trace
Experimental methodology

1. Source code
   - gcc --enable-logging
2. Server with logging enabled
   - Run experiments and collect traces
3. Raw traces
   - Localize security-sensitive operation
4. Relevant portions of traces
   - Compare traces with “diff” and “∩”
5. Pruned traces
Dynamic mining: Results

Each fingerprint localized to within 126 resource accesses
Limitations of dynamic mining

Security-sensitive operations

*Input Event*
- Create
- Destroy
- Copy
- Paste
- Map

Source Code

Runtime trace

1. Incomplete: False negatives
2. High-level description needed
3. Operations are manually induced
Outline

- Motivation
- Problem
- Solution
  - Fingerprints
  - Dynamic fingerprint mining
  - Static fingerprint mining [ICSE’07]
Static fingerprint mining

Security-sensitive operations

Input_Event
Create
Destroy
Copy
Paste
Map

Source Code

Output: Candidate Fingerprints

Cmp xEvent->type == KeyPress

Resources

• Window
• xEvent
Problem definition

- **R**: Set of resource accesses
  - *Read/Set/Cmp* of *Window/xEvent*
- **E**: Set of entry points into the server
- **Goal**: Find fingerprints using **R** and **E**

Not given an *a priori* description of security-sensitive operations
Straw-man proposal I

Each resource access in $R$ is a fingerprint

- Finest level of granularity
- $\textbf{Cmp} \ x\text{Event}->\text{type} \ == \ \text{KeyPress}$
- $\textbf{Read} \ \text{Window}->\text{firstChild}$
- $\textbf{Read} \ \text{Window}->\text{nextSib}$
- $\textbf{Cmp} \ \text{Window} \neq 0$
Problem with this proposal

Difficult to write and maintain policies at this level of granularity

- \textit{Cmp} \ xEvent->type == KeyPress
- \textit{Read} \ Window->firstChild
- \textit{Read} \ Window->nextSib
- \textit{Cmp} \ Window \neq 0
Straw-man proposal II

Each API in \( E \) is a fingerprint

- Coarsest level of granularity
  - \textit{Call} \texttt{MapSubWindows}
  - \textit{Call} \texttt{MapWindow}

- Write policies allowing/disallowing the use of an API call
Problem with this proposal

Does not reflect actual resource accesses performed by API call

- **Call** `MapSubWindows`
  - Enumerates child windows and maps them to the screen

- **Call** `MapWindows`
  - Maps a window onto the screen
Our approach

Cluster resource accesses that always happen together

- Each API entry point implicitly defines a set of resource accesses
- Cluster resource accesses based upon the API entry points that perform them
Static analysis

- Extract resource accesses potentially possible via each entry point
- Example from the X server
  - Entry point: `MapSubWindows(…)`
  - Resource accesses:
    
    ```
    Set xEvent->type To MapNotify
    Set Window->mapped To True
    Read Window->firstChild
    Read Window->nextSib
    Cmp Window ≠ 0
    ```
## Resource accesses

<table>
<thead>
<tr>
<th></th>
<th>MapSub Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set <code>xEvent-&gt;type</code> To MapNotify</td>
<td>✔</td>
</tr>
<tr>
<td><code>S_{E}</code> Identify candidate fingerprints by clustering resource accesses</td>
<td></td>
</tr>
<tr>
<td><code>R_\epsilon</code></td>
<td></td>
</tr>
<tr>
<td><code>Read Window-&gt;nextSib</code></td>
<td>✔</td>
</tr>
<tr>
<td><code>Cmp Window ≠ 0</code></td>
<td>✔</td>
</tr>
</tbody>
</table>
## Concept analysis

<table>
<thead>
<tr>
<th>Instances</th>
<th>MapSub Windows</th>
<th>Map Window</th>
<th>Keyboard Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Set xEvent-&gt;type To MapNotify}</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>\texttt{Set Window=}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{Read Window}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{Read Window-&gt;nextSib}</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{Cmp Window ≠ 0}</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{Cmp xEvent-&gt;type==KeyPress}</td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>

**Comparison via hierarchical clustering**
Hierarchical clustering

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set xEvent-&gt;type To MapNotify</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Set Window-&gt;mapped To True</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Read Window-&gt;firstChild</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Read Window-&gt;nextSib</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Cmp Window ≠ 0</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Cmp xEvent-&gt;type==KeyPress</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{MapSub Windows} & \quad \text{Map Window} & \quad \text{Keyboard Input} \\
\{A, B, C\}, \Phi & \quad \{C\}, \{6\} & \quad \Phi, \{1, 2, 3, 4, 5, 6\}
\end{align*}
\]
Mining candidate fingerprints

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>Set xEvent-&gt;type</code> To MapNotify</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td><code>Set Window-&gt;mapped</code> To True</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td><code>Read Window-&gt;FirstChild</code></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>Read Window-&gt;nextSib</code></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><code>Cmp Window ≠ 0</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><code>Cmp xEvent-&gt;type==KeyPress</code></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Cand. Fing. 1

Cand. Fing. 2

Cand. Fing. 3

{A, B, C}, Φ

{A, B}, {1, 2}

{A}, {1, 2, 3, 4, 5}

Φ, {1, 2, 3, 4, 5, 6}

{C}, {6}
## Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC</th>
<th>Cand. Fing.</th>
<th>Avg. Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>4,476</td>
<td>18</td>
<td>3.7</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>30,096</td>
<td>115</td>
<td>3.7</td>
</tr>
<tr>
<td>PennMUSH</td>
<td>94,014</td>
<td>38</td>
<td>1.4</td>
</tr>
</tbody>
</table>

![Bar chart showing size distribution for ext2, X server, and PennMUSH]
### Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified Security-sensitive ops</th>
<th>Candidate fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>22</td>
<td>115</td>
</tr>
</tbody>
</table>

Able to find **at least one fingerprint** for each security-sensitive operation.
### Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified Security-sensitive ops</th>
<th>Candidate fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>22</td>
<td>115</td>
</tr>
</tbody>
</table>

Identified as part of multi-year efforts \(v\) minutes, \(n\) hours.
Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified Security-sensitive ops</th>
<th>Candidate fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>22</td>
<td>115</td>
</tr>
</tbody>
</table>

- Associated 59 candidate fingerprints with security-sensitive operations
- Remaining are likely security-sensitive too
  
  Read Window->DrawableRec->width &
  Read Window->DrawableRec->height
Summary of contributions

Input_Event
Create
Destroy
Copy
Paste
Map

Can the client receive this Input_Event?
Lessons for the future

Modifying legacy code is non-trivial

- Modifications may break software
- Modifying executables is challenging

Low-overhead runtime system for policy enforcement on unmodified code
Lessons for the future

Soundness/completeness hard to achieve for C

- Type-safety violations the main problem

Provable guarantees with additional runtime checks?
Lessons for the future

**Difficult to automate failure handling**

- Failure handling is a crosscutting-concern
- Handling failure gracefully is the main challenge

**Aspect-oriented solution?**

**Checkpoint and rollback?**
“That’s all Folks!”
Errors in labeling traces (I)

<table>
<thead>
<tr>
<th></th>
<th>Open xterm</th>
<th>Close xterm</th>
<th>Move xterm</th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATE</strong></td>
<td>✔️</td>
<td></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>DESTROY</strong></td>
<td></td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>MAP</strong></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>UNMAP</strong></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>INPUTEVENT</strong></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Errors in labeling traces (I)

<table>
<thead>
<tr>
<th>Event</th>
<th>Open xterm</th>
<th>Close xterm</th>
<th>Move xterm</th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESTROY</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>MAP</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>UNMAP</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>INPUTEVENT</td>
<td></td>
<td></td>
<td>✔️</td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>

CREATE = Trace1 – Trace3
## Errors in labeling traces (II)

<table>
<thead>
<tr>
<th></th>
<th>Open xterm</th>
<th>Close xterm</th>
<th>Move xterm</th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CREATE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DESTROY</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>MAP</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>UNMAP</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>INPUTEVENT</strong></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Dealing with errors in labeling

- Missing labels from traces:
  - “∩” operation will not discard fingerprint
  - “diff” operation may erroneously eliminate a fingerprint

- Extra labels on traces:
  - May erroneously eliminate a fingerprint

- Trial-and-error
  - Relabel and recompute set-equations

- Empirically: tolerance of about 15% errors