Operations in Legacy Code

Reference monitor

Motivation for authorization policy enforcement

Static analyses to retrofit legacy code for authorization policy enforcement
- Mining security-sensitive operations via concept analysis
- Application to real-world servers
  - ext2 file system
  - X11 server
  - PennMUSH

Retrofitting for authorization
- Mandatory access control for Linux
- Linux Security Modules (Knight et al., '94)
- SELinux (Brumley et al., '94)
- Painstaking, manual procedure
  - Trusted X, Compartmented-mode workstation, X11/SELinux
  - Java Virtual Machine/SELinux
  - IBM Websphere/SELinux

Contributions

Outline
- Motivation
- Challenges
- Solution
- Case studies
- Conclusion

Undesirable information flow

Desirable information flow

Retrofitting the X server

- Operation request
- Response
- Subject: Alice
- Object: Bob_Window
- Security-sensitive operation: Input_Event

Vinod Ganapathy
University of Wisconsin-Madison
Trent Jaeger
Pennsylvania State University
Somesh Jha
University of Wisconsin-Madison

ICSE, Minneapolis, Minnesota
University of Wisconsin-Madison
ICSE 2007 Mining Security-Sensitive Operations in Legacy Code using Concept Analysis

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

Retrofitting lifecycle

Input_Event
Create
Destroy
Copy
Paste
Map

Security-sensitive operations
Source Code
Policy checks

Can the client receive this Input_Event?

Problems

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

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

Our contributions

Fingerprints
Matching

Matching Mining Fingerprints
[IEEE S&P 2006]

Outline

• Motivation
• Challenges
• Solution
  • Fingerprints
  • Matching fingerprints
  • Mining fingerprints
• Case studies
• Conclusion

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

• Input_Event :-
  Cmp xEvent->type == KeyPress

Examples of fingerprints

• Input_Event :-
  Cmp xEvent->type == MouseMove

• Enumerate :-
  Read Window->firstChild &
  Read Window->nextSib &
  Cmp Window # 0

Fingerprint matching

Perform Enumerate

Examples of fingerprints

• Input_Event :-
  Cmp xEvent->type == MouseMove

Examples of fingerprints

• Input_Event :-
  Cmp xEvent->type == KeyPress

Placing authorization checks

• X server function
  MapSubWindows

Examples of fingerprints

• Input_Event :-
  Cmp xEvent->type == MouseMove
Fingerprint mining

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

Straw-man solution I

Each resource access is a fingerprint
- Finest level of granularity
- \( \text{Cmp} \ x\text{Event} \rightarrow \text{type} == \text{KeyPress} \)
- \( \text{Read} \ \text{Window} \rightarrow \text{firstChild} \)
- \( \text{Read} \ \text{Window} \rightarrow \text{nextSibling} \)
- \( \text{Cmp} \ \text{Window} \neq 0 \)

Problem with this solution

Difficult to write and maintain policies at this level of granularity
- \( \text{Cmp} \ x\text{Event} \rightarrow \text{type} == \text{KeyPress} \)
- \( \text{Read} \ \text{Window} \rightarrow \text{firstChild} \)
- \( \text{Read} \ \text{Window} \rightarrow \text{nextSibling} \)
- \( \text{Cmp} \ \text{Window} \neq 0 \)

Straw-man solution II

Each API call is a fingerprint
- Coarsest level of granularity
- \( \text{Call} \ \text{MapSubWindows} \)
- \( \text{Call} \ \text{MapWindow} \)
- Write policies allowing/disallowing the use of an API call

Problem with this solution

Does not reflect actual resource accesses performed by API call
- \( \text{Call} \ \text{MapSubWindows} \)
  - Enumerates child windows and maps them to the screen
- \( \text{Call} \ \text{MapWindows} \)
  - Maps a window onto the screen

Our solution

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

Clustering resource accesses

<table>
<thead>
<tr>
<th>API Call</th>
<th>MapSub Windows</th>
<th>MapWindow</th>
<th>Keyboard Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set event-&gt;type to KeyPress</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Set Window-&gt;Mapped To None</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Read Window-&gt;FirstChild</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Read Window-&gt;nextSibling</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Cmp Window != 0</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Cmp event-&gt;type</td>
<td>=</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Concept analysis

Instances
- MapSub Windows
- MapWindow
- Keyboard Input

Clustering using concept analysis

\( \emptyset \), \( \{A\}, \{1, 2\} \)

Concept analysis

\( \{A, B, C\} \)
\( \{A\}, \{1, 2, 3, 4, 5\} \)
\( \{(A, B, C)\} \)
\( \{1, 2, 3, 4, 5\} \)
### Results on case studies

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified security-sensitive ops</th>
<th>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

### Motivation

- Identifying security-sensitive operations in legacy code is crucial for maintaining system integrity.

### Challenges

- Legacy code is complex and hard to analyze manually.
- Scalability and automation are critical for large codebases.

### Case studies

- Motivation and challenges are illustrated with case studies.

### Solution

- Concept analysis and fingerprinting techniques are applied.

### Results on case studies

<table>
<thead>
<tr>
<th>Software</th>
<th>LOC</th>
<th>Fingerprints</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>30,106</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>

### Summary

**Static approach to retrofit legacy code for authorization policy enforcement**

- Fingerprints
- Fingerprint mining with concept analysis
- Results:
  - mined fingerprints for security-sensitive operations in ext2, X server, and PennMUSH

### Mining Security-Sensitive Operations in Legacy Code

- Vinod Ganapathy
  - vg@cs.wisc.edu
- David King
  - dhking@cse.psu.edu
- Trent Jaeger
  - tjaeger@cse.psu.edu
- Somesh Jha
  - jha@cs.wisc.edu