Toward Automated Authorization Policy Enforcement

Vinod Ganapathy
vg@cs.wisc.edu

Trent Jaeger
tjaeger@cse.psu.edu

Somesh Jha
jha@cs.wisc.edu

March 1st, 2006
Second Annual Security-enhanced Linux Symposium
Baltimore, Maryland
Introduction

• SELinux helps meet information-flow goals

• Expressive access-control policy language

• Security-enhanced operating system
Security-aware Applications

• Need for security-aware applications

• Can we build applications that can enforce mandatory access control policies?
Security-aware Applications

• Need for security-aware applications

<table>
<thead>
<tr>
<th>Server</th>
<th>Request</th>
<th>Allowed?</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Yes/No</td>
<td>Yes/No</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Security-aware Applications

• Need for security-aware applications

• Our work: How to build security-aware applications?

• Focus is on mechanism, not policy
Motivating Example

Remote Client: Alice

Alice

Local

X Server
Motivating Example

Remote Client: Alice

Alice

Bob

X Server

Remote Client: Bob
Motivating Example

Remote Client: Alice

Remote Client: Bob

Keyboard input

Malicious client can snoop on input violating Alice’s confidentiality
Motivating Example

Remote Client: Alice

Remote Client: Bob

Malicious client can alter settings on other client windows
Motivating Example

Remote Client: Alice

Remote Client: Bob

X Server

No mechanism to enforce authorization policies on client interactions
Motivating Example

Remote Client: Alice

X Server

Remote Client: Bob

Goal of the Security enhanced X server project \cite{Kilpatrick et al., 2003}

Keyboard input

Disallowed
Need for Security-awareness

- More examples: user-space servers
  - Samba
  - Web servers
  - Proxy and cache servers
  - Middleware

- Common features
  - Manage **multiple clients** simultaneously
  - Offer **shared resources** to clients
  - Perform services on **behalf of their clients**
Main Claim

To effectively meet security-goals, all applications managing shared resources must be made security-aware.
Focus of our work

- How to build security-aware applications?
- Focus is on **mechanism**, not policy
  - Can use tools like Tresys’ SELinux Policy Management Toolkit
Security-aware Applications

Our work:
Tool support to retrofit legacy servers for authorization policy enforcement

• Retrofit existing, legacy code
  – Linux Security Modules project [Wright et al., 2002]
  – Security-enhanced X project [Kilpatrick et al., 2003]
  – Privilege separated OpenSSH [Provos et al., 2003]
Our Work

- Tools to analyze and retrofit legacy code
- Two case studies:
  - Retrofitting the X server [IEEE S&P 2006]
  - Retrofitting Linux [ACM CCS 2005]
Main Goal

**Main challenge:** Where to place reference monitor hooks?
Authorization Policies

• Access-control matrix [Lampson’71]

<table>
<thead>
<tr>
<th></th>
<th>/etc/passwd</th>
<th>/usr/vg/a.out</th>
<th>/var/log</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>r/w</td>
<td>r/w/x</td>
<td>r/w</td>
</tr>
<tr>
<td>vg</td>
<td>r/w/x</td>
<td>r</td>
<td></td>
</tr>
</tbody>
</table>

• Three entities: \langle subject, object, operation \rangle
  – Subject (user or process)
  – Object (resource, such as file or socket)
  – Security-sensitive operation (access vectors)
Main Goal

- Analysis techniques to find where server performs security-sensitive operations
Key Insight: Fingerprints

- Each security-sensitive operation has a fingerprint
- Intuition: Denotes key code-level steps to achieve the operation
Examples of Fingerprints

- Three access vectors from SELinux
  - **DIR_WRITE** :-
    - Set inode->i_ctime &
    - Call address_space_ops->prepare_write()
  - **DIR_RMDIR** :-
    - Set inode->i_size TO 0 &
    - Decrement inode->i_nlink
  - **SOCKET_BIND** :-
    - Call socket->proto_ops->bind()
Examples of Fingerprints

- Access vectors for the X server

  - **WINDOW_MAP:-**
    - Set `WindowPtr->mapped` TO TRUE &
    - Set `xEvent->type` TO MapNotify

  - **WINDOW_ENUMERATE:-**
    - Read `WindowPtr->firstChild` &
    - Read `WindowPtr->nextSib` &
    - Compare `WindowPtr ≠ 0`
Key Insight: Fingerprints

• How to **find** fingerprints?
• How to **use** fingerprints to place hooks?
Using Fingerprints: An Example

- X server function `MapSubWindows`

```c
MapSubWindows(Window *pParent, Client *pClient) {
    xEvent event;
    Window *pWin;
    ...
    pWin = pParent->firstChild; ...
    for (;pWin != 0; pWin=pWin->nextSib) {
        pWin->mapped = TRUE;
        ...
        event.type = MapNotify;
    }
}
```
Examples of Fingerprints

• Access vectors for the X server

  • \texttt{WINDOW\_MAP}:–
    - \texttt{Set} WindowPtr->mapped \texttt{TO TRUE} \&
    - \texttt{Set} xEvent->type \texttt{TO MapNotify}

  • \texttt{WINDOW\_ENUMERATE}:–
    - \texttt{Read} WindowPtr->firstChild \&
    - \texttt{Read} WindowPtr->nextSib \&
    - \texttt{Compare} WindowPtr \neq 0
Using Fingerprints: An Example

• X server function **MapSubWindows**

```c
MapSubWindows(Window *pParent, Client *pClient) { 
    xEvent event; 
    Window *pWin; 
    ... 
    pWin = pParent->firstChild; ... 
    for (; pWin != 0; pWin = pWin->nextSib) { 
        pWin->mapped = TRUE; 
        ... 
        event.type = MapNotify; 
    } 
}
```

Performs **Window_Map**
Examples of Fingerprints

• Access vectors for the X server

• \texttt{WINDOW\_MAP:–}
  – \textit{Set} WindowPtr->mapped \textbf{TO TRUE} &
  – \textit{Set} xEvent->type \textbf{TO MapNotify}

• \texttt{WINDOW\_ENUMERATE:–}
  – \textit{Read} WindowPtr->firstChild &
  – \textit{Read} WindowPtr->nextSib &
  – \textit{Compare} WindowPtr ≠ 0
Using Fingerprints: An Example

• X server function **MapSubWindows**

```c
MapSubWindows(Window *pParent, xEvent event, Window *pWin;
...
pWin = pParent->firstChild; ...
for (; pWin != 0; pWin = pWin->nextSib) {
  // Code to map window on screen
  pWin->mapped = TRUE;
  ...
  event.type = MapNotify;
}
```

Performs `Window_Enumerate`
Using Fingerprints

• Fingerprints located using static analysis
• Key advantage: statically find all locations where fingerprints occur
• Can add hooks to all these locations
Adding Hooks: An Example

• X server function `MapSubWindows`

```c
MapSubWindows( Window * pParent, Client * pClient ) {
    xEvent event;
    Window * pWin;
    // Code to enumerate child windows
    avc_has_perm( pClient, pParent, WINDOW_ENUMERATE);
    pWin = pParent->firstChild; ... 
    for ( ; pWin != 0; pWin = pWin->nextSib ) { 
        // Code to map window on screen
        avc_has_perm( pClient, pWin, WINDOW_MAP );
        pWin->mapped = TRUE;
        ... 
        event.type = MapNotify;
    }
}
```
Key Insight: Fingerprints

- How to **find** fingerprints?
- How to **use** fingerprints to place hooks? ✔
Finding Fingerprints

• Using analysis of runtime traces

• Key Insight:
  – If server does a security-sensitive operation its fingerprint must be in the trace

• Example:
  – Get X server to perform `WINDOW_MAP`

\[
\begin{align*}
\text{Set } & \text{WindowPtr->mapped TO TRUE} \\
\text{Set } & \text{xEvent->type TO MapNotify}
\end{align*}
\]
Finding Fingerprints

• Main challenge:
  – Locating fingerprints in the runtime trace

• Key insight:
  – Compare several runtime traces

Trace 1: Server does not perform \texttt{WINDOW\_MAP}
Finding Fingerprints

- Main challenge:
  - Locating fingerprints in the runtime trace
- Key insight:
  - Compare several runtime traces

```
Set WindowPtr->mapped TO TRUE
Set xEvent->type TO MapNotify
```

“DIFF”

Trace 2: Server does not perform WINDOW_MAP
Key Insight: Fingerprints

- How to find fingerprints?
- How to use fingerprints to place hooks?
Results

• Retrofitted version of X server
• Fingerprint-finding technique is effective:
  – Fewer than 10 functions to be examined to write fingerprints
  – In comparison, each trace exercises several hundred distinct X server functions
• Details in upcoming IEEE S&P 2006 paper
Examples of fingerprints

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINDOW_CREATE</td>
<td><strong>Call</strong> CreateWindow</td>
</tr>
<tr>
<td>WINDOW_DESTROY</td>
<td><strong>Call</strong> DeleteWindow</td>
</tr>
<tr>
<td>WINDOW_UNMAP</td>
<td><strong>Set</strong> xEvent-&gt;type TO UnmapNotify</td>
</tr>
<tr>
<td>WINDOW_CHSTACK</td>
<td><strong>Call</strong> MoveWindowInStack</td>
</tr>
<tr>
<td>WINDOW_INPUTEVENT</td>
<td><strong>Call</strong> ProcessPointerEvent, <strong>Call</strong> ProcessKeybdEvent</td>
</tr>
</tbody>
</table>
Slide to take home

- Goal: Placing authorization hooks in servers
- Key insight: Security-sensitive operations have fingerprints
  - Finding fingerprints: Using “diff” of runtime traces
  - Placing hooks: By statically locating fingerprints
Questions?

Toward Automated Authorization Policy Enforcement

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

http://www.cs.wisc.edu/~vg
http://www.cse.psu.edu/~tjaeger
http://www.cs.wisc.edu/~jha