A security sensitive operation is a distinct policy operation that is performed within code. Our hypothesis is that security sensitive operations leave fingerprints on protected data structures by reading or writing to them.

We model fingerprints as sets of structure member accesses—READs or WRITEs of protected data structures. Using static analysis on the legacy codebase, we determine what possible structure member accesses each API function.

Next, we use concept analysis, a hierarchical clustering technique, to organize these structure member accesses into distinct security-sensitive operations, resulting in a set of candidate fingerprints which can be refined into fingerprints by domain-specific constraints.

Finally, we place hooks into the old code, ensuring that every structure member access is mediated by the appropriate policy operation.

Experimental Results

We ran our static analysis on three distinct servers.

- The ext2 filesystem, included in the Linux Kernel
- The main dispatch loop of the X Windows server
- PennMUSH, the server for a multi-user online game

Source code analysis was done with a module written in CIL (C Intermediate Language), which uses plugins written in Objective Caml to perform source-code analysis on C programs.

It took about about a half hour of manual work to check whether or not each of the candidate fingerprints was security-interesting.

<table>
<thead>
<tr>
<th>Server</th>
<th>Lines of Code</th>
<th>Fingerprints</th>
<th>Avg. Fingerprint Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>4,476</td>
<td>18</td>
<td>3.36</td>
</tr>
<tr>
<td>X Server</td>
<td>30,096</td>
<td>115</td>
<td>3.76</td>
</tr>
<tr>
<td>PennMUSH</td>
<td>94,014</td>
<td>38</td>
<td>1.42</td>
</tr>
</tbody>
</table>

(1) Read addressspace->host
(2) Read ext2dir->reclen
(3) Write 0 To ext2dir->inode
(4) Read inode->mtime
(5) Read inode->u->ext2inode->info->i_start_lookup
(6) Write 1 To inode->u->ext2inode->info->i_start_lookup

A lattice generated by Concept Analysis for ext2. Security-interesting nodes are marked in red.

Future Work

A large amount of work remains in the area of retrofitting legacy systems for security. For example, it is necessary to improve our static analysis such that our results can scale better to even larger servers such as the Linux Server. What role can domain-specific and domain-independent constraints play in improving these results?

The security model that we use is heavily based on the model of structure member accesses: can we automatically mine richer policy from code and gain better guarantees?