Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable

Richard Ta-Min, Lionel Litty and David Lie
Department of Electrical and Computer Engineering
University of Toronto
Improving OS Security

• Hardware improves very quickly, but software changes at a slower pace:
  – Commodity software systems today still based on old legacy code

• Legacy code was never designed with security in mind:
  – Very few people used computers for sensitive operations
  – Most people did not even own computers, very few had high bandwidth internet connections

• Getting rid of legacy systems is impractical:
  – Too much effort is invested in commodity software

Should find ways to secure commodity software
Operating System Insecurity

- Commodity OS contain a lot of privileged code:
  - Gives attackers many opportunities to take control of OS

Should allow applications to control how they trust the OS
Our Solution: Proxos

Proxy Operating System (Proxos) routes system calls based on rules set by developer

David Lie
OIT METIS Security Seminar
Specifying System Call Routing Rules

• Proxos routes system calls based on the name of the resource, and the type of resource being accessed:

```c
DISK:("/etc/shadow", priv_fs)
...
priv_fs = {
    .open = priv_open,
    .close = priv_close,
    .read = priv_read,
    .write = priv_write
}
```

Example Routing Rules

• Rules link a method table (priv_fs) with name of resource (/etc/shadow)

• Method table has pointers to system call handlers in the private OS

• Resources not named in the rules are routed to commodity OS by default

Interface is partitioned into accesses to sensitive and non-sensitive resources
Starting a Private Application

- Hypercall added to VMM for host process to start private app:
  - Private VM system calls executed same host that started it

Commodity OS doesn’t have to trust private app any more than it trusts a regular app

David Lie

OIT METIS Security Seminar

Routing System Calls

System calls routed to commodity OS using RPC’s:
- During startup, a shared memory region between the commodity OS and Proxos is created
Proxos Prototype

- Proxos associates method tables with file descriptors:
  - When application open a resource, Proxos compares name with rules and binds methods to the descriptor
  - Methods are used in subsequent calls on the descriptor
Prototype Artefacts

• Proxos currently has artefacts that require minor porting for applications:
  – Single address space so forked code must be serialized
  – Not safe to dynamically load libraries from commodity OS, all code must be statically linked
  – No multi-threading support

• We ported three applications to Proxos:
  1. SSH server
  2. Web browser
  3. SSL web server
Application: SSH Server

- Proxos allows applications to have access to commodity OS, but isolated sensitive resources at the same time. Ex SSH Server:
  - Sensitive data such as user passwords and the host key stored in private OS
  - All network packets decrypted in private app before sent to command shell
Application: Graphical Web Browser

- Graphical Web Browser:
  - Sensitive user data and user interface is isolated from commodity OS
  - Browser can save downloaded files and invoke file viewers on the commodity OS
Application: Apache & SSL

- Server Application: Apache & SSL extension
  - Private signing key isolated from commodity OS
  - To maintain performance, minimize private VM startup/shutdown by making host process persistent
Evaluation: Effort to Port Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>Routing Rules (LOC)</th>
<th>App Modifications (LOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Browser</td>
<td>53</td>
<td>22</td>
</tr>
<tr>
<td>SSH Server</td>
<td>35</td>
<td>108</td>
</tr>
<tr>
<td>Apache &amp; OpenSSL</td>
<td>28</td>
<td>556</td>
</tr>
<tr>
<td>Glibc</td>
<td>--</td>
<td>218</td>
</tr>
</tbody>
</table>

- Two tasks required to port applications:
  1. Identify sensitive resources and write routing rules
  2. Modify applications to account for differences between Proxos and commodity OS
- Routing rules are local to each private application:
  - Adding a new application does not affect rules of other apps
Evaluation: Performance Overhead

- Overheads are dominated by forwarded system calls:
  - System calls experience 4 context switches
  - An additional context switch is caused by an artefact of our VMM configuration
Microbenchmarks

- Ran LMBench suite:
  - Minimal context switch time on our machine is 2.88 µs
  - Expected 5 context switches gives 14.4 µs
  - TLB misses resulting from context switches increase cost

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Linux (µs)</th>
<th>Proxos (µs)</th>
<th>Overhead (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL System Call</td>
<td>0.37</td>
<td>12.88</td>
<td>12.51</td>
</tr>
<tr>
<td>fstat</td>
<td>0.57</td>
<td>14.28</td>
<td>13.71</td>
</tr>
<tr>
<td>read</td>
<td>0.45</td>
<td>13.51</td>
<td>13.06</td>
</tr>
<tr>
<td>write</td>
<td>0.42</td>
<td>13.24</td>
<td>12.82</td>
</tr>
<tr>
<td>stat</td>
<td>8.76</td>
<td>25.98</td>
<td>17.22</td>
</tr>
<tr>
<td>open &amp; close</td>
<td>14.57</td>
<td>47.18</td>
<td>32.61</td>
</tr>
</tbody>
</table>
Application Benchmark

- Linear relationship: forwarded system calls is the dominant source of overhead
- Overheads overall are modest:
  - SSH experiences 6% slowdown and Apache even has a slight speedup

Linear Regression:

\[ y = 15.7 \times 10^{-6} x + 0.72 \]

\[ r = 0.92 \]
Conclusions

- By partitioning system call interface and routing system calls:
  - Can isolate sensitive resources from commodity OS
  - Still allow application to use commodity OS resources
  - Applications do not lose functionality

- Porting effort is reasonable:
  - Routing rules are short and on the order of 10’s of LOC
  - Rules are local to each application

- Performance:
  - Performance overheads are reasonable
  - Largely due to context switches and VM startup/shutdown
  - Some overheads can be removed by modifying application