Inline Reference Monitor

<table>
<thead>
<tr>
<th>App</th>
<th>Monitor</th>
</tr>
</thead>
</table>

- Move monitor into App
  - Share address space
  - Must protect integrity of monitor
  - Advantages
    - Lower overhead
    - Can monitor more internal state and actions of application
    - Strictly move powerful than external monitor
  - Challenges
    - Monitor integrity
    - How to do it
    - Complete mediation

SFI

Virtual memory

- Untrusted plugins
- Trusted

Scenario
- Untrusted plugin to trusted code
- Lots of RPC calls
- Goal: restrict untrusted code to its own memory + RPC

Rewrite memory references

<table>
<thead>
<tr>
<th>Original</th>
<th>rewrite (checking)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ld %r0, %r5</td>
<td>Mov %r30, %r5</td>
</tr>
<tr>
<td></td>
<td>mov %r31, %r30</td>
</tr>
<tr>
<td></td>
<td>And %r31, %r31, mask</td>
</tr>
<tr>
<td></td>
<td>Cmp %r31, segid</td>
</tr>
<tr>
<td></td>
<td>Jne ABORT</td>
</tr>
<tr>
<td></td>
<td>Ld %r0, %r30</td>
</tr>
</tbody>
</table>

Attack – jump straight to ld
- Dedicate register : %r30
- All loads/stores use %r30
- All moves to %r30 followed by check
- No SIGNALS

Another version of rewrite (forcing)
Mov %r30, %r5
And %r30, %r30, NSegmask
Or %r30, %r30, segid
Ld %r0, %r30

Jmp %r5 → mov %r30, %r5
And %r30, %r30, mask
Or %r30, %r30, segid
Jmp %r30

SFI RPC
- untrusted module can only jump into its own segment

Untrusted segment

Untrusted1 code
Launch pad
Gates
Untrusted2 code
Launch pad
Trusted launchpad
R gates

- Only escape is via launchpad
- Code in launchpad jumps to trusted code attacks
- Could jump into middle of trusted function
- Goal : untrusted code can only jump to specified locations
Launchpad

Untrusted code can safely jump anywhere in launchpad

Untrusted1 → launchpad → gates → reset register → untrusted2

SFI RPC
- copy args
- 2 jumps
- dedicated registers

OS RPC
- copy args: from caller to kernel
- save caller state
- load caller state
- OS overhead

OS RPC: ~200 micro second
SFI RPC: ~1 micro second