Safety in Systemtap
Discussion Notes

Brad Chen
Intel Corporation
20 April 2005
Outline

• Goals
• Feature List
• Design Overview
• Possible Enhancements
• Open Questions
Goals

Systemtap should be:

• crash-proof
• easy to program/debug
• easy to trust
• at least as safe as comparable systems on other platforms

These are goals, not requirements.

Systemtap should have “escape” to disable certain safety features as required for kernel debugging.
Feature Review

• Instruction restrictions
  – division by zero
  – illegal instructions
  – privileged instructions

• Control flow restrictions
  – infinite loops
  – recursion
  – kernel subroutines

• Memory bug protection
  – array bounds errors
  – invalid pointer errors
  – heap memory bugs

• Memory restrictions
  – memory read/write restrictions

• Version alignment

• End-to-end safety

• Separate safety policy from mechanism

Note: Checks applied to compiled script only. Runtime assumed safe.
## Design Overview

<table>
<thead>
<tr>
<th></th>
<th>Language Design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Language</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>insmod checks</td>
</tr>
<tr>
<td></td>
<td>Runtime Checks</td>
</tr>
<tr>
<td></td>
<td>Memory Portal</td>
</tr>
<tr>
<td></td>
<td>Static Validator</td>
</tr>
</tbody>
</table>

### Language Design Implementation

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Static Validation</th>
<th>Runtime Checks</th>
<th>Insmod Checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>infinite loops</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>recursion</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>division by zero</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>array bounds errors</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>invalid pointer errors</td>
<td>x</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>heap memory bugs</td>
<td>x</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>illegal instructions</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>privileged instructions</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>memory read/write restrictions</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>memory execute restrictions</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>version alignment</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>end-to-end safety</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>separate policy from mechanism</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Possible Enhancements
Static Validator

- Disassemble .ko before loading
- Check unrecognized code for conformance to safety rules
  - instruction restrictions
  - control flow restrictions
  - memory reference restrictions
- Recognize runtime and accept as safe
- Caveat: may need to restrict binaries to make them checkable
  - optimization flags
Static Validator Demonstration
Memory Portal

- A special-purpose interpreter
- Policy is provided independently of script
- Portal validates memory references with respect to policy, accept or reject
- Checks applied to compiled script only. Runtime assumed safe
- Check data and code memory references

- Optionally, static checker could verify that portal is being used
Memory Portal Policy Examples

- Systemtap default
  - all reads okay
  - no external writes
  - no external calls

- Guru mode
  - all reads okay
  - all writes okay
  - external calls okay

- UID protection
  - restrict reads by UID
  - no external writes
  - no external calls

- Script-specific policy
  - permit writes to a list of data structures or address range
  - permit calls to a list of kernel subroutines

Note: Checks applied to compiled script only. Runtime assumed safe.
Open Questions

- How to position the script validator
- Separation of safety policy from mechanism
  - Do we want to use a memory portal?
  - If not, do we want to make a different plan?
- Do tapset authors need the same safety features as script authors?
  - use of C and asm
  - external calls
  - native libraries