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.

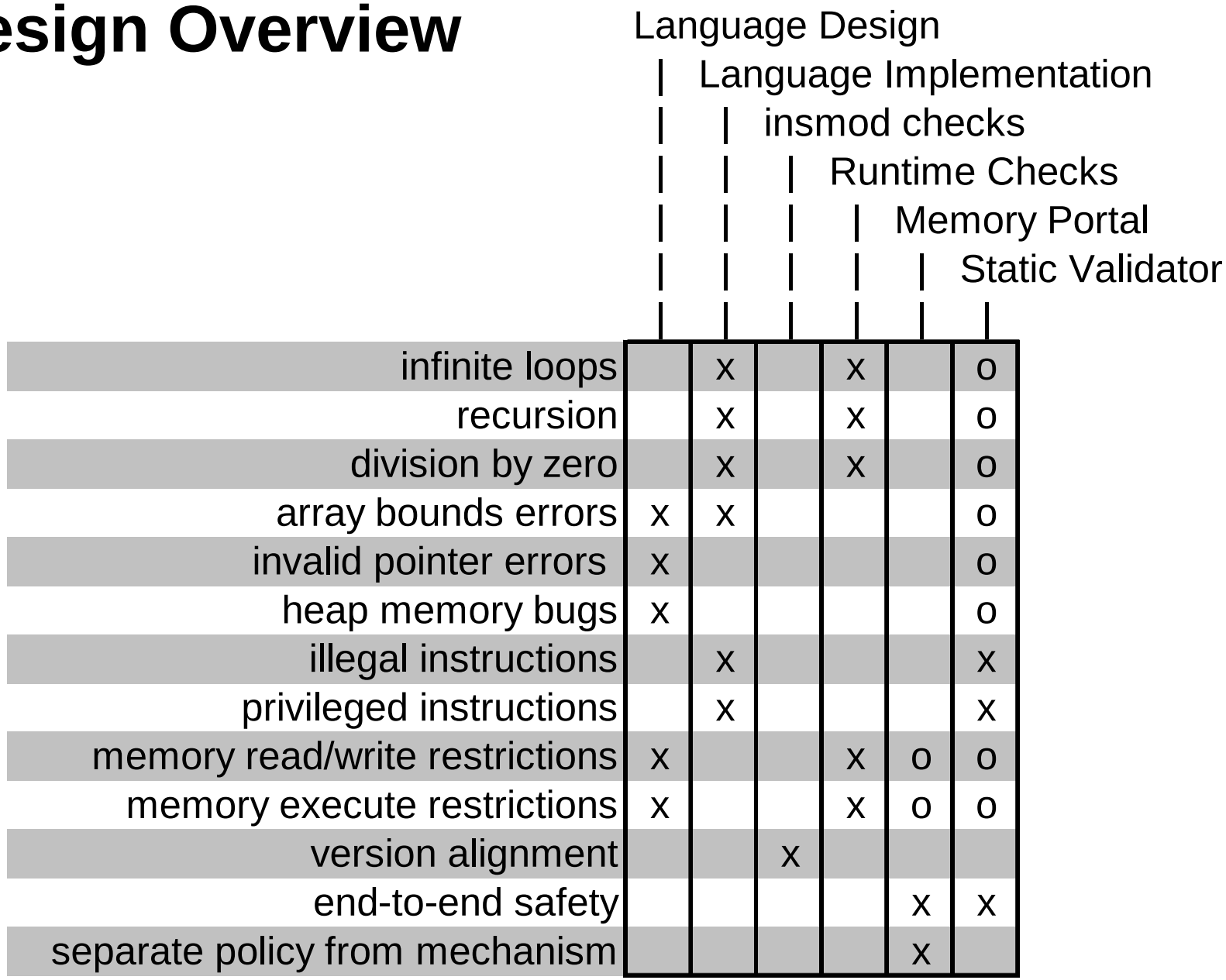
Systemtap should have “escape” to disable certain safety features as required for kernel debugging.

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



	Language Design	Language Implementation	insmod checks	Runtime Checks	Memory Portal	Static Validator
infinite loops		x		x		o
recursion		x		x		o
division by zero		x		x		o
array bounds errors	x	x				o
invalid pointer errors	x					o
heap memory bugs	x					o
illegal instructions		x				x
privileged instructions		x				x
memory read/write restrictions	x			x	o	o
memory execute restrictions	x			x	o	o
version alignment			x			
end-to-end safety					x	x
separate policy from mechanism					x	

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