Trusted and trustworthy

Part of your system is trusted if its failure can break your security.
Thus, OS is almost always trusted.
Real question: is it trustworthy?
Distinction not universally observed: trusted boot, Trusted Solaris, etc.

Trusted (I/O) path

How do you know you’re talking to the right software?
And no one is sniffing the data?
Example: Trojan login screen
Or worse: unlock screensaver with root password
Origin of “Press Ctrl-Alt-Del to log in”

Minimizing trust

Kernel → microkernel → nanokernel
Reference monitor concept
TCB size: measured relative to a policy goal
Reference monitor ⊆ TCB
But hard to build monitor for all goals

How to gain assurance

Use for a long time
Testing
Code / design review
Third-party certification
Formal methods / proof

Evaluation / certification

Testing and review performed by an independent party
Goal: separate incentives, separate accountability
Compare with financial auditing
Watch out for: form over substance, misplaced incentives

Orange book OS evaluation

Trusted Computer System Evaluation Criteria
D. Minimal protection
C. Discretionary protection
   - C2 adds, e.g., secure audit over C1
B. Mandatory protection
   - B1, B2, B3: stricter classic MLS
A. Verified protection
Common Criteria

- International standard and agreement for IT security certification
- Certification against a protection profile, and evaluation assurance level EAL 1-7
- Evaluation performed by non-government labs
- Up to EAL 4 automatically cross-recognized

Common Criteria, Anderson’s view

- Many profiles don’t specify the right things
- OSes evaluated only in unrealistic environments
  - E.g., unpatched Windows XP with no network attacks
- “Corruption, Manipulation, and Inertia”
  - Pernicious innovation: evaluation paid for by vendor
  - Labs beholden to national security apparatus

Formal methods and proof

- Can math come to the rescue?
- Checking design vs. implementation
- Automation possible only with other tradeoffs
  - E.g., bounded size model
- Starting to become possible: machine-checked proof

Proof and complexity

- Formal proof is only feasible for programs that are small and elegant
- If you honestly care about assurance, you want your TCB small and elegant anyway
- Should provability further guide design?

Some hopeful proof results

- seL4 microkernel (SOSP’09 and ongoing)
  - 7.5 kL C, 200 kL proof, 160 bugs fixed, 25 person years
- CompCert C-subset compiler (PLDI’06 and ongoing)
- RockSalt SFI verifier (PLDI’12)

Outline

- OS trust and assurance
- Announcements intermission

Note to early readers

- This is the section of the slides most likely to change in the final version
- If class has already happened, make sure you have the latest slides for announcements