e. Least privilege; above with root

9. Practise: detecting writing
   a. Integrity check files a la binaudit, tripwire; go through signature block
   b. LOCUS approach: encipher program, decipher as you execute.
   c. Co-processors: checksum each sequence of instructions, compute checksum as you go; on difference, complain

Notes for February 19, 1997

1. Hello
   a. Turn back midterms. mean was 70.
   b. When HW 2 returned (by Friday, I hope!) I'll also hand out a note saying how you're doing.
2. Puzzle of the day
   a. Key problem: how do you know it's really the student taking the test? Cameras, keystroke identification, etc.
3. Malicious logic
   a. Quickly review Trojan horses, viruses, bacteria
   b. Logic Bombs
   c. Worms (Schoch and Hupp)
4. Review trust and TCB
   a. Notion is informal
   b. Assume trusted components called by untrusted programs
5. Ideal: program to detect malicious logic
   a. Can be shown: not possible to be precise in most general case
   b. Can detect all such programs if willing to accept false positives
   c. Can constrain case enough to locate specific malicious logic
   d. Can use: writing, structural detection (patterns in code), common code analyzers, coding style analyzers, instruction analysis (duplicting OS), dynamic analysis (run it in controlled environment and watch)
6. Best approach: data, instruction typing
   a. On creation, it's type "data"
   b. Trusted certifier must move it to type "executable"
   c. UNIX idea: executable bit is "certified as executable" and must be set by trusted user (Duff)
7. Practise: blocking writing to communicate information or do damage
   a. Limit writing (use of MAC if available; show how to arrange system executables)
   b. Isolation
   c. Quarantine
8. Trust
   a. Untrusted software: what is it, example (USENET)
   b. Check source, programs (what to look for); C examples
   c. Limit who has access to what
   d. Your environment (how do you know what you're executing); UNIX examples