Notes for November 29, 1999

1. Greetings and Felicitations!
2. Puzzle of the Day
3. ORCON (Originator Controlled; Graubert)
   a. Document/information can be passed on with approval of originator; real world justification is that originator of document trusts recipients not to release documents which they should not.
   b. Untrusted subject *x* marks object *O* ORCON on behalf of organization *X* and indicates it is releasable to subjects acting on behalf of organization *Y*.
   not releasable to subjects acting on behalf of other organizations without *X*'s permission
   any copies made have the same restriction
   c. DAC: can't do this as the restriction would not copy over (*y* reads *O* into *C*, puts its own ACL on *C*)
   d. MAC: separate category with*O*, *x*, *y*. *y* wants to read *O*, copy to *C*; MAC means *C* has same category as *O*, *x*, *y*, so can't give *z* access to *C*.
   Say a new organization *w* wants to provide data in *B* to *y* but not to be shared with *x* or *z*. Can't use *O*'s category. Hence you get explosion of categories.
   Real world parallel: individuals are "briefed" into a category and those represent a formal "need to know" policy that is standard across the entity; ORCON has no central clearinghouse to categorize data; originator makes rules.
   e. Solution?
   owner of object can't change ACL's relationship with object (MAC characteristic)
   on copy, ACL is copied as well (MAC characteristic)
   access control restrictions can be tailored on a subject/object basis (DAC characteristic)
4. Malicious logic
   a. Quickly review Trojan horses, viruses, bacteria; include animal and Thompson's compiler trick
   b. Logic Bombs, Worms (Schoch and Hupp)
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 (duplicating 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. Duff's idea: executable bit is "certified as executable" and must be set by trusted user
7. Practise: 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; least privilege
   d. Your environment (how do you know what you're executing); UNIX examples
8. 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