Outline for March 10, 2003

Reading: text, §15.3–15.4, 22.1–22.5, 22.7

Discussion Problem

The PGP secure mailing system uses both RSA and a classical cipher called IDEA. When one installs PGP, the software generates two large (512 bits or so) numbers, to produce a modulus of 1024 bits. Such a number is too large to be factored easily. The private and public keys are generated from these quantities. The private key is enciphered with a classical cipher using a user-supplied pass phrase as the key. To send a message, a 128-bit key is randomly generated, and the message enciphered using IDEA with that key; the key is enciphered using the recipient's public key, and the message and enciphered key are sent.

1. If you needed to compromise a user's PGP private key, what approaches would you take?
2. It's often said that PGP gets you the security of a key with length 1024. Do you agree?

Outline for the Day

1. Lock and Key
   a. Associate with each object a lock; associate with each process that has access to object a key (it's a cross between ACLs and C-Lists)
   b. Example: use cryptography. X object enciphered with key K. Associate an opener R with X. Then:
      OR-Access: K can be recovered with any D_i in a list of n deciphering transformations, so
         $R = (E_1(K), E_2(K), ..., E_n(K))$ and any process with access to any of the D_i's can access the file
      AND-Access: need all n deciphering functions to get K: $R = E_1(E_2(...E_n(K)...))$
   c. Types and locks
2. MULTICS ring mechanism
   a. MULTICS rings: used for both data and procedures; rights are REWA
   b. (b_1, b_2) access bracket - can access freely; (b_3, b_4) call bracket - can call segment through gate; so if a's access bracket is (32,35) and its call bracket is (36,39), then *assuming permission mode (REWA) allows access*, a procedure in:
      rings 0-31: can access a, but ring-crossing fault occurs
      rings 32-35: can access a, no ring-crossing fault
      rings 36-39: can access a, provided a valid gate is used as an entry point
      rings 40-63: cannot access a
   c. If the procedure is accessing a data segment d, no call bracket allowed; given the above, *assuming permission mode (REWA) allows access*, a procedure in:
      rings 0-32: can access d
      rings 33-35: can access d, but cannot write to it (W or A)
      rings 36-63: cannot access d
3. Malicious logic
   a. Quickly review Trojan horses, viruses, bacteria; include animal and Thompson's compiler trick
   b. Logic Bombs, Worms (Schoch and Hupp)
4. 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)
5. 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
6. 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

7. 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