6.857 Rivest
L19.1 4/15/09

Admin: Quizzes back today (end of class)
stats:
See TA's this week re project
PS #5 out later today

Outline:

- ☐ Thompson's "Reflections on Trusting Trust" (1984 Turing Award)
- ☐ Software bugs
- ☐ hardware bugs
  - Shamir's μproc mpy bug (& related smartcard attack) ↓
- ☐ keyloggers

- Ken Thompson's "Reflections on Trusting Trust" (1984)
  example of nasty malware: can't even find it by looking at source code (!)

let L = login program L(pw): [if check(pw)
then allow_login()
else reject()]

evil login program L'(pw): [if pw = "34NQ74B"
or check(pw)
then allow_login() else reject()]

but: someone may notice source has been modified

so: attacker can also modify compiler (!)
Let C = standard compiler

evil compiler C'(x) = if x = L then output C(L')
else output C(x)

now source for L is left alone, but source for compiler changed; it may be noticed.

- so doubly-evil compiler:

$C''(x) =$ if $x = L$ then output $C(L')$
else if $x = C$ then output $\underline{C(C'')}$ ← note self-reference!
else output $C(x)$

Attacks leaves sources as $L, C, \ldots, x$
but binaries as $C(L'), C(C''), \ldots, C'(x)$

all sources look clean!
situation is <u>stable</u>: recompiling any program yields same binary!

<u>Ouch!!!</u>

Moral (Thompson): "You can't trust any code you did not totally create yourself!"

- Thompson's paper a good example of how you can "bug" the software to an adversary's advantage.

- What else could be done by adversary, in this vein?
  - kleptography
  - random # generator with only 35 bits of entropy
  - file system (send files away)
  - configuration errors
  - network (send copy of all packets elsewhere...)
  - keylogger (sends copies of all keystrokes to adversary)
  - botnet

  this Q is too easy!

- Adversary's objectives:
  - Bug should introduce vulnerability that is easy for adversary to exploit.
  - Bug should be hard to discover without detailed examination of code or hardware.
  - Deniability: Bug should look like innocent mistake rather than malicious attack.
  - Bug should be hard for others to exploit, even if they know about it.
  - Bug should be hard to fix.

(how does Thompson measure up here?)

What about putting bugs in hardware?

- in CPU:
  - undocumented opcodes
  - " behavior
    (e.g. if register R1 contains 0xFA23CB1C then following instruction is executed with no protections)
  - system maintenance mode bug
    (what happens at boot time?)

- Shamir's bug:
  - targets crypto (PK) implementations (bignum)
  - bug in 64-bit mpy: $r \cdot s \neq r \cdot s \pmod{2^{64}}$
    for two particular values r, s (random-looking)
  - hard to discover, even though in all μproc
    2^{-128} chance of hitting r, s per mpy
    2^{40} processors on planet
    2^{30} seconds (≈ 30 years)
    * 2^{30} mpys/sec
    2^{100} mpys tried; $\leq 2^{-28}$ of hitting bad one

RSA: $n = p \cdot q$
$S = M^d$ (reduce M mod p, or mod q, first)

CRT: $S_p = M^{d_p} \pmod{p}$ $d_p = d \bmod p-1$
$S_q = M^{d_q} \pmod{q}$ $d_q = d \bmod q-1$

$S = a \cdot S_p + b \cdot S_q \pmod{n}$
$a \equiv 1 \bmod p$ $a \equiv 0 \bmod q$
$b \equiv 0 \bmod p$ $b \equiv 1 \bmod q$

How to utilize bug, as adversary?

Suppose $p < \sqrt{n} < q$ (p, q unknown to adversary)

Let $M = \sqrt{n}$ with least sig 2 64-bit words replaced by r, s

Then $M \bmod p \neq M$
but $M \bmod q = M$

Computing $M^d \bmod n$ using CRT

$d = d_k d_{k-1} \cdots d_1 d_0$ $\quad d_0 = \text{lsb}$

$A \leftarrow 1$
$x \leftarrow M$
for $i = 0, 1, \ldots, k$
  if $d_i = 1$ then $A \leftarrow A \cdot x \pmod p$ (mod q) ← depending on which part of CRT we are doing
  $x \leftarrow x^2 \pmod p$ (or mod q)

↖ needs to compute r·s gets it wrong mod q !

S is correct answer (both mod p & mod q)
S' is " mod p but not mod q

$\gcd(S - S', n) = p$!

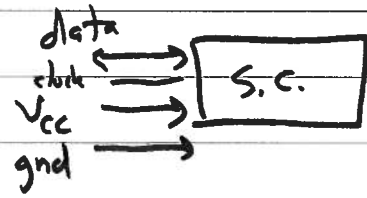
(Note: easy to get S, using self-reducibility (blinding))

Fix? (no CRT!) (randomize) (check answer! with PK)

Related attack: (not "bug", but related...)

power glitch or timing glitch

smart card



smart card does RSA comp. using CRT

short clock cycle during mod q part of CRT can have same effect as bad mpy, ⇒ bad result mod q

power spike

6.857 Rivest
L19.8 4/15/09

More on keyloggers

- Could be
  - in OS
  - hypervisor ("blue pill")
  - hardware
    - in line
    - in keyboard
    - modified keyboard software
  - acoustic?
- How do they get data out?
  - pw causes data to be regurgitated
  - wireless
- How to defeat?
  - keyboard on screen (use mouse) ?
  - move focus in and out of text entry area?
  - use one-time password (SecurID)
    - changes every 60 secs
    - PRG based on AES
    - synchronization on RSA...