Admin: Quizzes back today (end of class)

stat:

See TAs this week re project
PS #5 out later today

Outline:

- Thompson's "Reflection on Trusting Trust" (1984 ACM Award)
- Software bugs
- Hardware bugs
  - Shamir's mpoc mp7 bug (related smarthud attack)
- Keyloggers
Ken Thompson's "Reflections on Trusting Trust" (1984)

Example of nasty malware: can't even find it by looking at source code (!)

```plaintext
let L = login program
L(pw):
  if check(pw)
    then allow login()
  else reject()
```

```plaintext
evil login program
L'(pw):
  if pw = "3YNQ748"
    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

```plaintext
evile 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:

\[ C''(x) = \begin{cases} 
C(L') & \text{if } x = L \\
C(C') & \text{if } x = C \\
C(x) & \text{else}
\end{cases} \]

\[ \text{note self-reference!} \]

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

all sources look clean!

situation is stable: recompiling any program yields same binary!

Ouch!!

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...)

- This Q is too easy!

  - keylogger (sends copies of all keystrokes to adversary)
  - botnet

- Adversary's objectives:
  - Bug should introduce **vulnerability** that is easy for adversaries 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 0xFAR3GB1C
    then following instruction is executed with no protections)
  - system maintenance mode bug
    (what happens at boot time?)

- Shamir's bug:
  - target crypto (PK) implementations (bignum)
  - bug in 64-bit mpjs: \( r \cdot s = r \cdot s \pmod{2^{128}} \)
    for two particular values \( r, s \) (random-looking)
  - hard to discover, even though 100% of procs
    \( 2^{128} \) chance of hitting \( r, s \) per mpj
  - \( \frac{2^{40}}{2^{30}} \) processors on planet
  - \( 2^30 \) seconds (\( \approx 30 \) years)
  - \( \left( \frac{2^{30}}{2^{100}} \right) \) mps/sec

- RSA: \( n = p \cdot q \)
  \[ S = M^d \]
  \[ \text{CRT: } S_p = M^{d_p} \pmod{p} \quad d_p = d \pmod{p-1} \]
  \[ S_q = M^{d_q} \pmod{q} \quad d_q = d \pmod{q-1} \]
  \[ S = a \cdot S_p + b \cdot S_q \quad a = 1 \pmod{p} \quad a = 0 \pmod{q} \]
  \[ b = 0 \pmod{p} \quad b = 1 \pmod{q} \]
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 \( p, q \)

Then \( M \mod p \neq M \)

but \( M \mod q = M \)

Computing \( M^d \mod n \) using CRT

\[
d = d_k d_{k-1} \ldots d_1 d_0 \quad d_0 = \text{lsb}
\]

\[ A \leftarrow 1 \]

\[ x \leftarrow M \]

for \( i = 0, 1, \ldots, k \)

if \( d_i = 1 \) then \( A \leftarrow A \times (\mod p) \)

\[
x \leftarrow x^2 \mod p \ (\text{or } \mod q)
\]

\[ \text{needs to compute } r \cdot s\]

gets it wrong \( \mod q \)!

\( S \) is correct answer \( \mod p \& \mod q \)

\( S' \) 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

\[
\begin{array}{c}
\text{data} \\
\text{clock} \\
V_{\text{cc}} \\
\text{gnd}
\end{array} \rightarrow \boxed{\text{S.c.}}
\]

- Smart card does RSA comp. using CRT

- Short clock cycle during mod q part of CRT can have same effect as bad mpq, \( \Rightarrow \) bad result mod q

- Power spike
More on keyloggers

- Could be - in OS
  - hypervisor (blue"pill")
  - hardware - in line - acoustic?
    - in keyboard
      - modified keyboard/software
  - How do they get data out?
    - pw causes data to be regurgitated
    - wireless

- How to detect?
  - Keyboard on screen (use mouse)?
  - move focus in and out of text entry areas?
  - use one-time password (SureID)
    - changes every 40 secs
    - PRG based on AES
    - synchronization on RSue...