

G.857 Rivest
L19.1 4/13/11

Admin:

Outline: Malware

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

6.857 Rivest

L18.2 4/13/11

- 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 $L(pw)$: $\left[\begin{array}{l} \text{if check}(pw) \\ \text{then allow_login}() \\ \text{else reject}() \end{array} \right.$

evil login program $L'(pw)$: $\left[\begin{array}{l} \text{if } pw = "3YNQ74B" \\ \text{or check}(pw) \\ \text{then allow_login}() \text{ else reject}() \end{array} \right.$

but: someone may notice source has been modified

so: attacker can also modify compiler (!)

Let C = standard compiler

evil compiler $C'(x) = \left[\begin{array}{l} \text{if } x = L \text{ then output } C(L') \\ \text{else output } C(x) \end{array} \right.$

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

• so doubly-evil compiler:

C.857 Rivest
L19.3 4/13/11

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

Attacks treats sources as L, C, \dots, x
but binaries as $C(L'), C(C''), \dots, C(x)$

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!"

6.857 Rivest

L19.4 4/13/11

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

e 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 0xFA23GB1C 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 μ proc
 2^{-128} chance of hitting r, s per mpy

* $\left(\begin{array}{l} 2^{40} \text{ processors on planet} \\ 2^{30} \text{ seconds (x 30 years)} \\ 2^{30} \text{ mpy/sec} \\ 2^{100} \text{ mpy, tried; } \leq 2^{-28} \text{ of hitting, bad one} \end{array} \right.$

RSA: $n = p \cdot q$
 $S = M^d$

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

$d_p = d \pmod{p-1}$
 $d_q = d \pmod{q-1}$

reduce $M \pmod{p}$, or \pmod{q} , first

$$S = a \cdot S_p + b \cdot S_q \pmod{n}$$

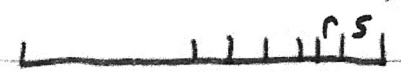
$$a \equiv 1 \pmod{p} \quad a \equiv 0 \pmod{q}$$

$$b \equiv 0 \pmod{p} \quad b \equiv 1 \pmod{q}$$

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 r, s



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

Computing $M^d \bmod n$ using CRT

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

~~for~~

$$A \leftarrow 1$$

$$x \leftarrow M$$

for $i = 0, 1, \dots, k$

$$\text{if } d_i = 1 \text{ then } A \leftarrow A \cdot x \pmod{p} \pmod{q}$$

$$x \leftarrow x^2 \pmod{p} \text{ (or mod } q)$$

depending on which part of CRT we are doing

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

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

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

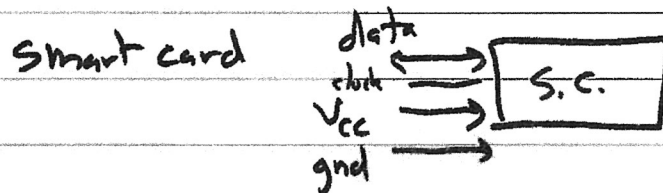
Fix? (no CRT!) (randomize) (check answer!)
with PK

6.857 Rivest

L19.7 4/13/11

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

power glitch or timing glitch



smart card does RSA comp. using CRT

short clock cycle during mod q part of CRT can have same effect as bad mpy, \Rightarrow bad result mod q
power spike

G.857 Rivest
L19.8 4/13/11

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 reencrypted
 - 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 40 secs
 - PRG based on AES
 - synchronization on BS4E...