CSE/EE 461 Lecture 25
Security Practice

Tom Anderson
tom@cs.washington.edu

Security Practice

- In practice, systems are not that secure
  - hackers can go after weakest link
    - any system with bugs is vulnerable
  - vulnerability often not anticipated
    - usually not a brute force attack against encryption system
  - often can't tell if system is compromised
    - hackers can hide their tracks
  - can be hard to resecure systems after a breakin
    - hackers can leave unknown backdoors

Password Attack/Response

- Moore's Law: enables large number of passwords to be checked very quickly
- Countermeasure
  - Delay password check for 1 second, so can't try them quickly
  - Need to delay both successful and unsuccessful password checks!
- Counter-countermeasure:
  - Observe network traffic; extract any packet encrypted in password; check various passwords offline
- Counter-counter-countermeasure:
  - Kerberos: don't use password to encrypt packets; instead use password to encrypt file containing shared key; use shared key to encrypt packets
- Counter-counter-counter-countermeasure: …

Kerberos Weaknesses

- Early versions of Kerberos had several security flaws
  - block cipher: allowed encrypted blocks to be replaced
    - A -> B (transfer $10 to Tom's account)
    - A -> B (transfer $1M to Wells Fargo)
    - solution: add encrypted CRC over entire message
  - used timestamps to verify communication was recent
    - time server communication not encrypted
    - get time from authentication server
  - Kerberos login program downloaded over NFS
    - NFS authenticates requests, but data is unencrypted
    - disallow diskless operation

802.11 Weaknesses

- Ports often installed behind the firewall
  - anyone can listen, send packets on intranet
- Weak encryption method
  - uses 40 bit key, 32 bit initial #
  - most implementations use same initial #, allowing dictionary, replay attacks
- Key management
  - single key used for all senders on a LAN
  - often disabled
- Uses parity instead of CRC for integrity
  - allows block replacements that maintain parity

Internet Worm

- Used the Internet to infect a large number of machines in 88
  - password dictionary
  - sendmail bug
    - default configuration allowed debug access
    - well known for several years, but not fixed
  - fingerd: finger tom@cs
    - fingerd allocated fixed size buffer on stack
    - copied string into buffer without checking length
    - encode virus into string!
- Used infected machines to find/infect others

Ping of Death

- IP packets can be fragmented, reordered in flight
- Reassembly at host
  - can get fragments out of order, so host allocates buffer to hold fragments
- Malformed IP fragment possible
  - offset + length > max packet size
  - Kernel implementation didn't check
- Was used for denial of service, but could have been used for virus propagation

TCP/DNS Hijacking

- Example: Mitnick
  - denial of service attack against system administrator
    - open large number of TCP connections
  - scan for open, idle TCP connections (e.g., rlogin, xwindows)
    - send bogus TCP packets to other end
    - e.g., to modify .rhosts to allow mitnick access
- Example: DNS cache poisoning
  - watch DNS cache for when it fetches new translation
    - e.g., for cnn.com
  - spoof reply to poison cache to point to bogus server

Netscape

- Used time of day to pick session key
  - easy to predict, break
- Offered replacement browser code for download over Web
  - four byte change to executable made it use attacker's key
- Buggy helper applications (ex: ghostview)
  - if web site hosts infected content, can infect clients that browse to it

Microsoft

- Browser runs Java, supposedly "safe"
  - random byte code generation found numerous bugs that caused crashes
  - many could be used to covertly insert viruses
- Email viruses: Melissa, etc.
  - Attachments can run code that is poorly sandboxed

Code Red/Nimda

- Dictionary attack of known vulnerabilities
  - known Microsoft web server bugs, email attachments, browser helper applications, …
  - used infected machines to infect new machines
- Code Red:
  - designed to cause machines surf to whitehouse.gov simultaneously
- Nimda:
  - Left open backdoor on infected machines for any use
  - Infected ~ 400K machines; approx ~30K still infected

Thompson Virus

- Ken Thompson self-replicating program
  - installed itself silently on every UNIX machine, including new machines with new instruction sets
- Aside: can you write a self-replicating C program?
  - program that when run, outputs itself
    - without reading any input files!
  - ex: main() { printf(“main () { printf(“main () …

Add backdoor to login.c

- Step 1: modify login.c

  A:

  ```
  if (name == "ken") {
      don't check password;
      login ken as root;
  }
  ```

- Modification is too obvious; how do we hide it?

Hiding the change to login.c

- Step 2: Modify the C compiler

  B:

  ```
  if see trigger {
     insert A into the input stream
  }
  ```

- Add trigger to login.c

  /* gobblygook */

- Now we don't need to include the code for the backdoor in login.c, just the trigger
  - But still too obvious; how do we hide the modification to the C compiler?

Hiding the change to the compiler

- Step 3: Modify the compiler

  C:

  ```
  if see trigger2 {
     insert B and C into the input stream
  }
  ```

- Compile the compiler with C present
  - now in object code for compiler
- Replace C in the compiler source with trigger2

Compiler compiles the compiler

- Every new version of compiler has code for B,C included
  - as long as trigger2 is not removed
  - and compiled with an infected compiler
  - if compiler is for a completely new machine: cross-compiled first on old machine using old compiler
- Every new version of login.c has code for A included
  - as long as trigger is not removed
  - and compiled with an infected compiler

Lessons

- Hard to resecure a machine after penetration
- Hard to detect if machine has been penetrated
- Any system with bugs is vulnerable

Soapbox

- Information = property
  - is it ok to break into a computer system if you don't intend to steal anything -- just to look around?

Course Topics

- Internet architecture
  - how a web request works, from click to display
    - DNS lookup, connection setup, request/response to server, IP routing, media access, wire signalling, …
  - end to end principle
- Link layer
  - Signal transmission
  - Checksums and CRC's
  - Media access (Ethernet)

Course Topics

- Routing (IP)
  - forwarding and addressing mechanics
  - link state and distance vector routing (OSPF)
  - interdomain routing (BGP)
  - server load balancing and NATs
- Transport (TCP)
  - ARQ and sliding window
  - Connection setup/teardown and flow control
  - Remote procedure call
  - Congestion control: RTT estimation and window size

Course Topics

- Services
  - DNS lookup, caching and replication
  - distributed cache coherence
- Multicast
  - forwarding, routing, retransmission, congestion control
- Real-time
  - scheduling and buffer management
  - resource reservations
- Security
  - encryption and why that's not enough

Internet Design Principles

- End to end principle
  - Expect failures to occur at every step, so end hosts must be ultimately responsible for error recovery
  - example: TCP checksum, sliding window
- Soft state
  - if possible, state should be recoverable after a failure
  - example: link state routing messages are resent periodically, whether needed or not
- Design for scalability
  - using backoff: Ethernet, TCP congestion control
  - using hierarchy: IP addresses, DNS, routing (BGP)
  - using neighbors: IGMP, multicast retransmissions

The Future: Reliability

- Internet has ~ 98-99% uptime
  - measured end to end: can two hosts communicate?
  - telephone network: 99.99% uptime
  - air traffic control: 99.999% uptime
- How do we build more reliable systems?
  - Internet effective at masking router/link failures
  - Remaining failures are operational mistakes, programming errors, malicious attacks
  - Need more robust protocol design methodology!