Chapter 19: Security

- The Security Problem.
- Password-Based Authentication.
- Sample Threats.
- Threat Monitoring.
- Firewalls.
- Authentication.
- Encryption.

The Security Problem

- Security must consider external environment of the system, and protect it from:
  - Unauthorized access.
  - Malicious modification or destruction.
  - Prevention of legitimate use.
- Easier to protect against accidental than malicious misuse.

Finding a Password

- If you know the user, try spouse's name or date of birth.
- *Brute force*: try all possible combinations of letters and numbers.
- *Dictionary attack*: try all words in a dictionary, alone or in combination.
- *Shoulder surfing*: look at the keyboard while user types password.
- *Keystroke recorder*: Internet cafe computers record all keystrokes.
- *Network sniffing*: computers normally discard network packets that are not addressed to them. But all packets on a LAN are electrical signals on the same wire. Hence any computer can listen to all traffic.
- *Time-based attack*: measure and analyze time it takes to reject a guess, and deduce how close to the true password the guess is; then try again.
- Steal password file while using system as a guest.

Protecting Passwords

- Use hard-to-guess passwords: long, non-words, initials, punctuation.
- Force short random delay after each unsuccessful authorization attempt.
- Encrypt network traffic.
- Store passwords in encrypted form (Unix `crypt()`).
- Make password file accessible only to administrator and special login program (running under administrator domain).
- Use one-time passwords: computer presents random challenge *c*, user computes and submits *f(c)*, computer computes *f(c)* and compares. Only user and computer know function *f*.
- Multiple-factor authentication: *f* is very complex, and varies with time. User uses calculator to compute *f(c)*, and calculator requires a PIN (Personal Identification Number) to do computation.

Sample Threats

- *Trojan horse*:
  - Bob composes a new attractive plugin for a text editor.
  - Sue, who works under Bob, uses the new plugin, and it works well...
  - ... except that it quitely copies every file Sue edits into a directory Bob owns.
  - When Sue edits a confidential letter to the CEO about Bob's bad management, Bob reads it before Sue sends it. Bob fires Sue.
- Trojan horse variation:
  - Bob writes a program that displays a full-screen dialog that looks identical to the Windows login screen.
  - Bob runs the program and leaves the Internet cafe without logging out.
  - Sue types her username and password in Bob's program.
  - Bob stores that information away, then his program executes a logout, and so Sue gets the (true) Windows login screen back.
  - Sue assumes she mistyped her password the first time, tries again, and logs in.
- Internet Trojan horse:
  - Bob buys the domain name `www.citibanc.com`, and sets up a web page that looks like Citibank's home page.
  - Sue mistypes the URL, logs onto Bob's page.
  - Bob steals Sue's password, then redirects to the true Citibank site.

Sample Threats (Cont.)

- *Trap door*:
  - Bob is hired by the Bank of Vietnam to write their account management software.
  - Bob writes it, and it works great, except that it has two trap doors:
    1. Bob's code optionally rounds amounts downwards, i.e. $1.995 becomes $1.99. The extra $0.005 is deposited to Bob's personal account.
    2. Bob's code includes a segment where he checks for a specific username and password against hard-coded constants instead of the regular password file. Bob logs in using that account to activate the previous code segment only on the day the bank calculates interest payments.
  - Imagine if Bob had written a compiler that added a trap door to every compiled program!

Sample Threats (Cont.)

- Stack and buffer *overflow*:
  - The Windows RPC server listens for requests by running the `listen()` procedure.
  - `listen()` receives a request to run an RPC call.
  - `listen()` calls a `getUserAndPassword()` procedure to receive the authorization information.
  - The caller sends a string longer than the string buffer allocated by `getUserAndPassword()`.
  - Windows doesn't check the string length, and so the return address from the call into `getUserAndPassword()` and back to `listen()` is overwritten with part of the string.
  - The string wasn't random: the overflow part pointed to itself as the return address and...
  - the rest of it was a malicious program.

Sample Threats (Cont.)

- *Worm*:
  - *Grappling hook* program sent to Bob's computer from Sue's via RPC stack overflow (or one of many other techniques, in general).
  - The hook is very small, and all it does is upload the main worm program from Sue's computer.
  - The main worm
    - connects to other computers via RPC to distribute the grappling hook, and then
    - deletes the Windows folder on Bob's hard drive.
- *Denial of Service* (DOS):
  - SCO (a US company) is funded by Microsoft to file lawsuits against companies that use Linux.
  - SCO angered some Linux fans, who release a DOS attack against its web site.
  - Fans write simple scripts that pretend to be browsers wanting to connect to SCO's web site...
  - but terminate the connection right after they make a request, and...
  - immediately issue another request, and another, and so on.
  - SCO's web site had been inaccessible for months.

Sample Threats (Cont.)

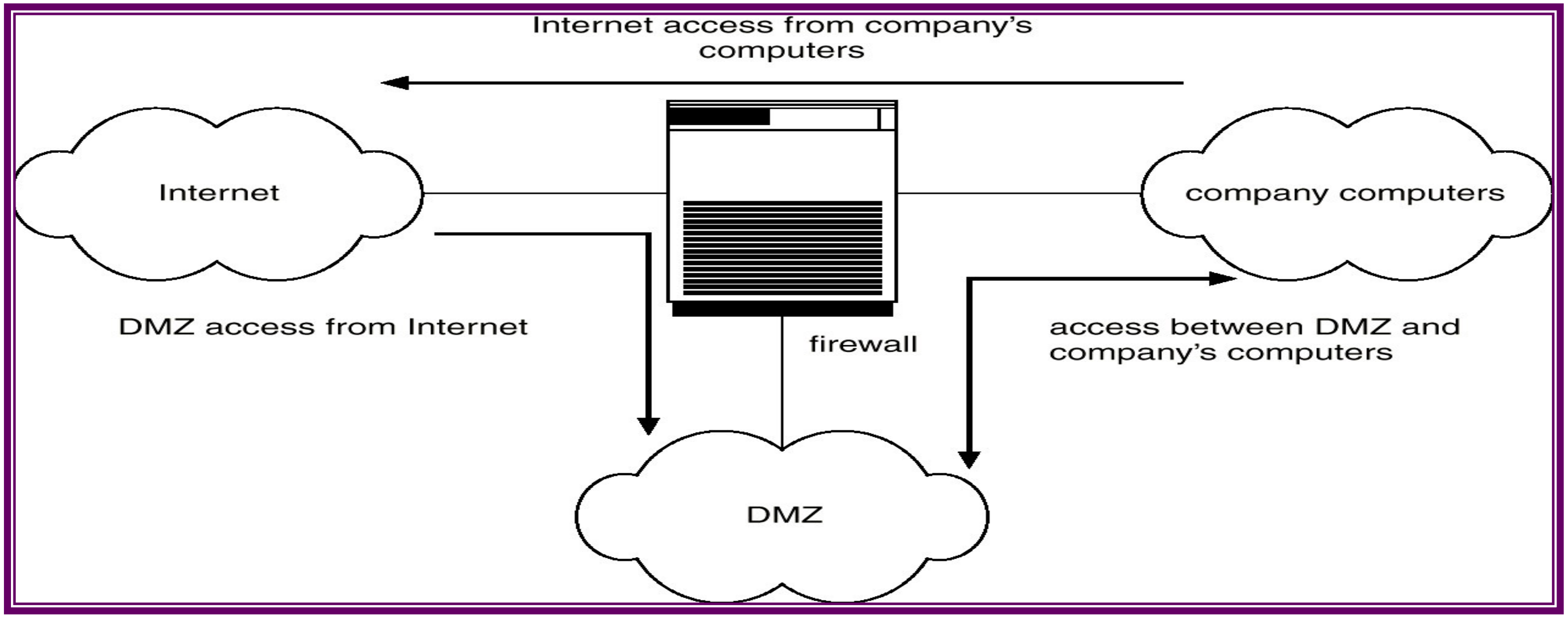
- *Virus*:
  - Like trap door, a virus is part of a useful program.
  - But a single virus can attach itself to many different programs: its operation is irrelevant from that of the infected program.
  - It simply inserts itself in the middle of regular program code.
  - When executed, it copies itself to other programs, and/or causes damage (choice usually based on number of times executed, or specific date, e.g. Michelangelo virus).
  - Programs and data not well separated any longer:
    - Word macros, Excel custom formulas are VB programs executed when document/spreadsheet is loaded.
    - Email attachments (e.g. the *love bug* virus).
    - Web pages (e.g. javascript that replaces *hosts* file, resolving `www.yahoo.com` to an ad site).

Threat Monitoring

- Maintain *audit log*: record the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.
- Check for suspicious patterns of activity in audit log, e.g. several incorrect password attempts may signal password guessing.
- Scan the system periodically for security holes; executed while the computer is relatively unused. Check for:
  - Short or easy-to-guess passwords.
  - Unauthorized setuid programs.
  - Unauthorized programs in system directories.
  - Unexpected long-running processes.
  - Improper directory protections.
  - Improper protections on system data files.
  - Dangerous entries in the program search path (Trojan horse).
  - Changes to system programs: compute checksum of each system program executable when installing them, and store the checksums off-line. Recompute and check against originals.

Firewalls

- Regular firewall is screen to prevent flames from an engine fire to reach the car’s passengers.
- LAN firewall: router that inspects and may reject network packets:
  - Placed between trusted T and untrusted U hosts.
  - It allows network messages to travel from T to U without restrictions.
  - Messages from U to T go through only if they are replies to earlier messages sent from T to U.
- DMZ (Demilitarized Zone):
  - All messages from U to T that are not replies go to DMZ computers.
  - DMZ contains web server, VPN server (which accesses T computers securely on behalf of authorized U hosts), etc.



Authentication

- Sue receives a message from Bob. How does Sue know whether Bob indeed sent it?
- Message Authentication Code (MAC):
  - ☞ Bob and Sue meet secretly and choose function *f(m)* to map every text message *m* into a number *f(m)*. *f(m)* is *collision-resistant*, like a hash, to map different messages into different numbers.
  - ☞ Bob sends his message *m* along with a number *a=f(m)*. *a* is the *authenticator*.
  - ☞ Sue receives the message, computes *b=f(m)* on her own, and makes sure *a==b*. If so, it's indeed from Bob.
  - ☞ In general, *f(m)* is one of a family of functions *F(k,m)* for a specific number *k*, the key. Bob and Sue just share a common secret key *k*.
- Digital Signatures:
  - ☞ RSA (see next slide).

Authentication: RSA

- No secret meeting needed: Bob has two keys:
  - Private key: Bob uses it to generate an authenticator *a* that accompanies every message *m* he sends.
  - Public key: Sue uses it along with *a* to check the authenticity of *m*'s sender (i.e. to make sure it's Bob).
- Specifics:
  - Private key: an integer *d*.
  - Authenticator generation (signature):

    $a = f(m)^d \bmod pq$

    f(m) maps text messages to numbers and is collision-resistant.

    p, *q* are two large primes.
  - Public key: the pair (*e*,*pq*) where *e* is such that

    $ed \bmod (p-1)(q-1) == 1.$
  - Verification:

    $a^e \bmod pq == f(m).$
  - Everybody knows *f(m)*, *e*, and *pq* but only Bob knows *d*. *p* and *q* are not individually known, so it's not easy to derive *d* from *e*.

RSA Example

- Private key: 3.
- Authenticator generation (signature):

  p=3, q=5, a=$f(m)^3$ mod 15.
- Public key: the pair (11,15) because 33 mod 2x4==1.
- Verification:

  a^{11} mod 15==$f(m)$.
- Check:
  - Assume $f(m)$=3.
  - Signature: 3^3 mod 15=27 mod 15=12.
  - Verification: 12^{11} mod 15=743008370688 mod 15==3.

Encryption

- Scramble clear text into *cipher* to send across an insecure line.
- A good encryption technique:
  - Makes it simple for authorized users to encrypt and decrypt data.
  - Depends not on the secrecy of the algorithm but on a parameter of the algorithm called *the encryption key*.
  - Makes it extremely difficult for an intruder who listens to encrypted traffic to deduce the encryption key.
- *Data Encryption Standard* (DES) substitutes characters and rearranges their order on the basis of an encryption key (56 bits) provided to authorized users via a secure mechanism. Scheme only as secure as this mechanism.
- Current NIST standard: the Rijndael ("Rain Doll") algorithm (Federal Information Processing Standards Publication 197).
- RSA can be used for encryption if
  - *f(m)* is the message itself,
  - *d* is public (i.e. anybody can compute *a*) and,
  - *e* is private (i.e. only intended recepient can reproduce *f(m)*).

Secure Sockets Layer (SSL)

- Certification authority (CA): company that issued a *certificate* for Bob and stores Bob's public key.
- Sue (her browser actually) has public key of CA.
- Sue connects to Bob's web site:
  - Bob sends Sue random number n_s and his certificate C.
  - Sue contacts CA, sends C, receives Bob's public key and verifies authenticity of CA reply via public key of CA.
  - Sue sends Bob a random p, encrypted with Bob's public key.
  - Bob receives p and decrypts it using his private key.
  - Bob and Sue both compute two DES keys and two MAC keys based on n_s and p, one pair for each direction of communication. DES and MAC keys are session-specific because n_s and p are random; keys are secure because p was transmitted securely.
  - To send a message, Bob/Sue computes the authenticator (using the MAC key) and encrypts it along with the message using the DES key.
  - To receive a message, Sue/Bob decrypts the incoming cipher using the DES key, and checks the authenticator using the MAC key.
- DES and MAC used instead of RSA because they are faster to encrypt/decrypt for every single message.