Problems

- Millions of computers are "infected" by malware of various kinds
- Any computer connected to the Internet is subject to attacks intended to subvert it to malicious use
- Denial-of-service attacks can shut down servers and even render whole parts of the network useless
- Credit Card numbers are (evidently) regularly stolen when they are carried over the network
Background

• The protocols used in the Internet were experimental in nature
  – Nobody knew how to make scalable public networks
  – The builders just wanted to make them work

• The participants in the early Internet experiments were mostly researchers
  – They trusted each other and wanted to share data

• It was expensive and difficult to connect to the Internet
  – A malicious computer connecting to the network was a low-probability occurrence
Background

• Internet Protocols were designed with security in mind
  – In fact, the IP specification (RFC 791) includes a Security Option, for carrying information related to security
  – Based on an entirely different security model (from what we use today)
    • Nothing to do with cryptographic mechanisms
    • Secure information was kept completely separate from everything else

• The world changed!
  – The assumptions under which the Internet Protocols were designed are no longer valid
    • Intentionally simple and easy to connect to the Internet
    • Easy to "snoop" others' packets
    • Broadband access lines ⇒ easy to concentrate "streams" of packets
Security Threats

• Machine Compromise (breakin)
  - Exploiting software weaknesses (usually bugs) to take over a machine
  - The compromised machine often becomes part of a network of such machines, used to do the attacker's bidding
    • zombie: a compromised machine that sits and waits to be told when to go into action
    • botnet: a network of compromised machines that can act in concert to attack a third party.

• Eavesdropping
  - "Snooping" traffic as it travels over a "wire" to obtain sensitive information
    • More often: over a wireless link
  - Often leads to compromise (e.g. snooping passwords)
Security Threats

• Impersonation
  - Attacker acts like someone else in order to gain information
    • Phishing web sites resemble a legitimate bank, (store, etc.) web site; Site collects "payment details" (account or credit card numbers) from the real site's customers
    • Phishing may also involve email – e.g. that looks like it's from the government (Internal Revenue Service, Immigration & Naturalization Service, ...)

• Denial-of-Service (DoS)
  - Overwhelming a server or other component so legitimate users cannot obtain service
  - Usually involves a botnet as source of traffic
    • 1000's of hosts send traffic at low rate → high-rate flood overwhelms router or server
Compromise via **Buffer Overflow**

- By far the most common method
- Exploits two things:
  - Characteristics of C language:
    - Strings are character arrays terminated by null (0) byte
    - No array bounds checking
    - Automatic (local) variables allocated on stack
  - Bad programming
    - Any information received from the outside world **must** be treated with suspicion! *Never assume!*
    - Canonical mistake: assume a string fits in a fixed-size buffer
      ```c
      void func(char *s) {
        char buf[BUFSIZE];
        strcpy(s,buf);    // buffer overflow!
        ...
      }
      ```
Principle of Buffer Overflow Attacks

Program Stack

- Stack grows downward
- Increasing addresses

- stack pointer
- Stack grows downward

- local variables
  - buf:
  - return addr
  - parameters
  - s: a$&*(!&...

stack frame for func
Principle of Buffer Overflow Attacks

Program Stack

Stack grows downward

Return value overwritten by string. Contents of string determine to where function returns!
Non-Cryptographic Solutions

- Defensive Programming
  - Eliminate software errors that open door for attacks
- Firewalls
  - Stop "undesirable" packets from reaching end systems
Defensive Programming

• Never, never, never assume anything about input from the outside world, regardless of whether it is:
  – typed
  – read from a file
  – received over the network

• Always, always, always verify that length does not exceed buffer size (or truncate so it does) when traversing input from outside
Firewalls

• A **mechanism** that enforces **policy** about what packets are allowed across a **boundary**
  - Example: router at the edge of a network
    • Permissible packets are **forwarded** into/out of the network
    • Other packets are silently **dropped**
  - Example: at the top of the IP layer, inside a host
    • Permissible packets passed up to higher/down to lower layer
    • Others dropped (possibly with notification – cf Windows)
Firewalls

- **A mechanism** that enforces **policy** about what packets are allowed across a **boundary**
  - Example: router at the edge of a network
    - Permissible packets are **forwarded** into/out of the network
    - Other packets are silently **dropped**
  - Example: at the top of the IP layer, inside a host
    - Permissible packets **passed up to higher/down to lower layer**
    - Others **dropped** (possibly with notification – cf Windows)
Filtering Rules

- Patterns for packet fields:
  - Source, destination IP address
  - Source, destination port
  - Protocol
  - [TCP] flags
- Action: allow, deny
- First matching rule is applied

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dest IP</th>
<th>Src port</th>
<th>Dest port</th>
<th>Protocol</th>
<th>Flags</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.163.*</td>
<td>*</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
<td>*</td>
<td>allow</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>21</td>
<td>TCP</td>
<td>*</td>
<td>deny</td>
</tr>
<tr>
<td>*</td>
<td>128.163.1.1</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
<td>*</td>
<td>allow</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
<td>*</td>
<td>deny</td>
</tr>
<tr>
<td>128.163.*</td>
<td>*</td>
<td>*</td>
<td>25</td>
<td>TCP</td>
<td>*</td>
<td>allow</td>
</tr>
</tbody>
</table>
Firewall Challenges

• Problem: ambiguity
  – Filter pattern only sometimes not enough to determine direction
  – Solution: inbound/outbound ruleset for each interface
  – Also: i/f-independent ruleset applied in the middle
Firewall Challenges

- Problem: policies have to be translated into filter patterns
  - Example:
    - "Allow incoming connections to Web server"
    - "Allow outgoing connections except to certain bad servers"
    - "Allow incoming connections to mail server, except from spammers"
  - A cumbersome and error-prone process
  - Errors can deny service or leave network wide open
  - Port numbers and addresses are terrible mechanisms for enforcing policies
Cryptographic Solutions

• Crypto Building Blocks
  – Block, Stream Ciphers: support confidentiality
  – Digital Signatures: support authenticity and integrity
  – Secure Hash functions: support integrity
  – Protocols: combine/use building blocks in a secure manner to provide services

• IPSec: security at the network layer

• SSL/TLS: security at the transport layer
Crypto Building Block: Block Cipher

- Block cipher is an invertible $e_K: \{0,1\}^B \rightarrow \{0,1\}^B$
  - $C = e_K(P)$, where $K = \text{key}$; $C = \text{ciphertext}$; $P = \text{plaintext}$
  - $P = d_K(C)$ \hspace{1cm} \text{Note: } d_K(e_K(P)) = P$
  - Security depends (only) on size of the keyspace (i.e. \# bits in key)

- Properties:
  - Given $C_0, C_1, C_2, \ldots$, \textit{computationally hard} to determine key $K$ or any plaintext $P_0, P_1, P_2, \ldots$
  - Given many ciphertext-plaintext pairs, hard to determine key

  \text{Note: } "\textit{computationally hard}" \text{ means no known method significantly better than brute-force (try all possibilities)"}
Crypto Building Block: Block Cipher

Examples:

- DES: (now obsolete) 64-bit blocks, 56-bit keys
  - US Standard for encryption, used in business 30+ years
- 3DES: DES used repeatedly
  - typical: $E_K(w) = e_{K_1}(d_{K_2}(e_{K_1}(w)))$
    where $e = DES$, $K=(K_1,K_2)$
  - This gives backward-compatibility with DES implementations by putting $K_1=K_2$
- AES: 128-bit blocks, key lengths up to 256 bits
  - Standard (NIST) replacement for DES
- Blowfish: 64-bit blocks, key lengths 32 to 448 bits
  - Invented by Bruce Schneier
- IDEA: 64-bit blocks, 128-bit keys
  - patented
Crypto Building Block: Stream Cipher

- Stream cipher maps a stream of bits (plaintext) to a different, apparently-random stream of bits (ciphertext)
  - Key determines mapping
  - In principle, the streams may be infinite
    - In practice, there is a (very high) limit on how much plaintext can be encrypted with a given key

- Properties:
  - Given ciphertext, computationally hard to determine key or plaintext

- Examples: RC4 (key size up to 2048)

- Note differences between block cipher and stream cipher:
  - Stream cipher allows plaintext to be encrypted one bit (or byte) at a time; block cipher requires minimum 1 block plaintext (i.e., padding)
  - Stream cipher has state: same plaintext encrypts to different ciphertexts depending on location in stream:
    - \texttt{themaninblack} $\rightarrow$ \texttt{xysemloixhgw}
Crypto Building Block: Secure Hash

• Also known as Message Digest or "one-way" function
  - Map arbitrary messages to concise "fingerprints"
  - Important for digital signatures, authentication

• Random mapping \( h: \{0,1\}^+ \rightarrow \{0,1\}^N \)
  - \( N \) = size of hash, typically at least 128 or more bits

• Properties:
  - Given \( y \), computationally hard to find \( x \) such that \( h(x) = y \)
  - Computationally hard to find any two \( x, x' \) such that \( h(x) = h(x') \)
  - Any change to \( x \) changes about half the bits (on average) in \( h(x) \)
Crypto Building Block: Secure Hash

- Probability of "Collision" (for a random function) depends on the square root of the size of the space
  - That is: for N-bit hash, have to examine about $2^{(N/2)}$ inputs to find a collision
  - "Birthday paradox" – given 23 people in a room, probability of two with same birthday is $> \frac{1}{2}$
    - \[
    \text{Prob[all } n \text{ birthdays different]} = \frac{365!}{365^n(365-n)!}\]

- Examples:
  - MD5 – 128 bits, **broken**, i.e. known $x$, $x'$ s.t. $\text{MD5}(x) = \text{MD5}(x')$
  - SHA-1 – 160 bits, **under suspicion** (flawed attack published)
  - SHA-256 – 256 bits, still OK
Crypto Building Block: HMAC

- **HMAC**: hashed *message authentication code*
  - Authentication of origin of message
  - Detection of tampering (modification) of message

- **Components**:
  - Message $m$
  - Secure hash function $h$ (e.g. SHA-1, MD5)
  - Secret $K$ shared between sender and verifier

- **Operation**:
  - Sender computes $x = h([K \oplus Pi] \cdot h([K \oplus Po] \cdot m))$, sends $m$, $x$
    - $Pi$, $Po = $ padding patterns
  - Verifier computes $x'$ same way, checks $x' = x$
    - Any change to $m$ changes the outcome

- **Note**: HMAC-MD5 is still considered secure, even though MD5 is broken w.r.t. collision-resistance
Building Block: Digital Signatures

- **Idea:** verify authenticity of origin of data
  - Prove that a message (pile of bits) was signed by a particular principal (the signer)

- **Pieces:**
  - Signer's key $k$
  - Verifier's key $K$
    - Could be same as signer's key (e.g. in HMAC)
  - Signing function, produces signature $\sigma = \text{sign}(m, k)$
  - Verifying (boolean) function:
    - $\text{verify}(m, \sigma, K): \text{true iff } \sigma = \text{sign}(m, k)$

- **Properties**
  - Given $m$, hard to forge $\sigma$ without knowledge of $k$
Problem: Key Management

- For most applications, communicating parties need to share a key
  - Note: key should be unguessable
- If Alice and Bob have not met before, how can they establish a shared key?
  - Need a secure channel to set up the key!
- One solution: centralized Key Server
  - Key Server KS is a Trusted Third Party (TTP)
  - Every principal has a secure channel to KS (secret key)
    - Created, e.g. by going to a specific location and getting a disk
- When Alice wants to communicate with Bob, she asks KS for a "ticket" to Bob
  - KS creates $key_{AB}$
  - KS computes $T = \{Alice, timestamp, key_{AB}\}_{key_{Bob}}$
  - KS sends $\{Alice, Bob, T, timestamp, key_{AB}\}_{key_{Alice}}$ to Alice
  - Alice sends $\{"Hi from Alice", nonce\}_{key_{AB}} T$ to Bob
  - Bob sends $\{"Hi Yourself", nonce+1\}_{key_{AB}}$ to Alice
Public-Key Cryptography

• Idea: create a key for each principal, split into two parts
  – "Public" part $K$ – known to everybody
  – "Private" part $k$ – known only to that principal

• To communicate with Alice:
  – Encrypt message with Alice's public key
  – Only Alice can decrypt because only she knows the private key

• Signing (if supported):
  – Alice encrypts something with her private key
  – Anybody can decrypt using Alice's public key!

• New problem:
  – How do I get Alice's public key?
Crypto Building Block: Diffie-Hellman Key Exchange

- Allows Alice and Bob to establish a shared secret key without prior contact!
- How it works:
  - All participants agree on parameters $g$ and $p$
    - $p$ (the modulus) is a specially-chosen (large) prime number
    - $g$ is primitive mod $p$
  - Alice:
    - Chooses a private number $a$
    - Computes $A = g^a \mod p$, her public number (public key)
  - Bob:
    - Chooses a private number $b$
    - Computes $B = g^b \mod p$, his public number (public key)
  - Alice sends $A$ to Bob, Bob sends $B$ to Alice
  - Alice computes $X = B^a \mod p$, Bob computes $Y = A^b \mod p$
  - But $X = (g^b)^a = g^{ab} = (g^a)^b = Y$ (all modulo $p$)!
Diffie-Hellman Key Exchange

Bob: $b, g, p$
$B = g^b \mod p$

$X = A^b \mod p$

$["Hi Alice"]_X$

Alice: $a, g, p$
$A = g^a \mod p$

$X = B^a \mod p$

$["Hello Bob"]_X$

Security basis is the **Discrete Log Problem**: Given $N$, find $n$ such that $N = g^n \mod p$
Problem: Man-in-the-Middle Attack

How does Alice know that number she received really came from Bob?

Bob: b, g, p
B = g^b mod p

Mel:
C = g^c mod p
D = g^d mod p

Alice: a, g, p
A = g^a mod p

Y = C^b mod p
Z = A^d mod p
Y = B^c mod p
Z = D^a mod p

["Hi Alice"]_Y
["Hello Bob"]_Y

["Hi Alice"]_Z
["Hello Bob"]_Z
RSA Encryption

• Numbers:
  - Modulus $n$
  - Public exponent $e$
  - Private exponent $d$
• Properties:
  • $n = pq$, where $p$ and $q$ are large primes (300-500 bits)
    - $p$ and $q$ must be kept secret!
  • $ed = 1 \mod (p-1)(q-1)$
    - Note: $\varphi(n) = (p-1)(q-1)$ [Euler $\varphi$-function]
• Property that makes it work:
  - $M^{ed} = M \pmod n$, for any $M$
• Security:
  - Given $n$, $e$, hard to find $d$, $p$, or $q$
  - Basis is DLP and hardness of factoring
RSA Encryption

• To send M to Alice (only), given her public key:
  - Compute \( M' = M^e \mod n \), send it to Alice
  - Only Alice can compute \( M'^d = M^{ed} = M \)

• Alice can sign a document D:
  - First compute a hash \( x = h(D) \)
  - Compute signature \( s = x^d \mod n \)
  - To verify \((D,s)\): check that \( s^e = h(D) \)
Public Key Infrastructure

• Needed: a secure method of verifying the binding between public key and person
  – How do I get the public key of somebody over the net in a reliable way?

• Public-Key Certification
  – A few trusted authorities certify bindings
    • By issuing (and signing) public-key certificates
    • Example:
      CA Alpha says "Alice's public key is (n,e)". Expires 12/18/06 signed, CA Alpha

  – Public keys of certification authorities are well-known
    • E.g., built into web browsers!
Public Key Infrastructure

• **X.509**: standard for public key certificates
  – Defines the layout of information in certificate

• **Problem**: To what should public keys be bound?
  – IP Address? Domain name? Some other name?
  – None of these is a reliable indicator of personal identity!
  – X.509 standard uses X.500 Identifiers – not globally managed/used
    • Usual practice is to use domain names, IP addresses in "auxiliary information" – also part of certificate
    • But there is no standard for this! Open to spoofing, phishing, etc.
Secure Sockets Layer Protocol (v3)

- **Origin**: Netscape/Microsoft/Netscape
  - Transport Layer Security (TLS) is similar (but not compatible) IETF standard

- **Architecture**: add on top of a TCP connection
  - Implemented in a library
  - Communicating parties must agree out-of-band, **in advance**, to use it!
SSL Typical Operation

Alice (client)

Hi, my random # is \( R_{\text{Alice}} \)

Bob here. My random # is \( R_{\text{Bob}} \)

cert: ["Bob's RSA public key is \( K_{\text{Bob}} \)]_{\text{verisign}}

Choose random \( S \); compute

\[ K = \text{PRF}(S, R_{\text{Alice}}, R_{\text{Bob}}) \]

\[ \{S\}K_{\text{Bob}} \]

\{keyed hash of msgs sent so far\}\( X \)

\{keyed hash of msgs sent so far\}\( Y \)

\{data encrypted using keys derived from \( K \}\}

Note: Client typically authenticated by server in application-specific manner over the secure channel.
SSL With (Optional) Client Authentication

Alice (client)

Choose random S;
compute K = PRF(S, RAlice, RBob)

Z, Z': keyed hash of messages sent so far

Bob (server)

Hi, my random # is RAlice

Bob here. My random # is RBob
Please send your certificate.
here's mine: ["Bob's RSA public key is KBob"]_verisign

Verisign

my cert: ["Alice's DSA public key is KAlice"]_Thawte

Thawte

\{S\}KBob

verify signature
compute K = PRF(S, RAlice, RBob)

\{Z\}Alice

\{Z'\}Y

\{data encrypted using keys derived from K\}
Other SSL/TLS Features

• Resumable Sessions
  – Client can resume a session with a server via an abbreviated handshake
    • Requires that both sides maintain state while disconnected

• Either side can initiate rekeying at any time
  – Keys should be changed occasionally because they "wear out"