Focus For Today’s Lecture

• Sufficient background in networking to then explore security issues in next 4 lectures
  – Networking = the Internet

• Complex topic with many facets
  – We will omit concepts/details that aren’t very security-relevant
  – We’ll mainly look at IP, TCP, DNS and DHCP

• Networking is full of abstractions
  – Goal is for you to develop apt mental models / analogies
  – ASK questions when things are unclear
    – (but we may skip if not ultimately relevant for security, or postpone if question itself is directly about security)
Key Concept #1: **Dumb Network**

- Internet design: interior nodes ("routers") have no knowledge* of ongoing connections going through them

- Not: how you picture the telephone system works
  - Which internally tracks all of the active voice calls

- Instead: the postal system!
  - Each Internet message ("packet") self-contained
  - Interior "routers" look at destination address to forward
  - If you want smarts, build it "end-to-end"
  - Buys simplicity & robustness at the cost of shifting complexity into end systems

* Today’s Internet is full of hacks that violate this
Key Concept #2: **Layering**

- Internet design is strongly partitioned into layers
  - Each layer relies on services provided by next layer below …
  - … and provides services to layer above it

- Analogy:
  - Consider structure of an application you’ve written and the “services” each layer relies on / provides

```
Code You Write
Run-Time Library
System Calls
Device Drivers
Voltage Levels / Magnetic Domains
```

Fully isolated from user programs
Internet Layering ("Protocol Stack")
Layer 1: Physical Layer

Encoding bits to send them over a single physical link, e.g. patterns of voltage levels / photon intensities / RF modulation
Layer 2: Link Layer

Framing and transmission of a collection of bits into individual messages sent across a single “subnetwork” (one physical technology)

Might involve multiple physical links (e.g., modern Ethernet)

Often technology supports broadcast transmission (every “node” connected to subnet receives)
Layer 3: (Inter)Network Layer

- Application
- Transport
- (Inter)Network
- Link
- Physical

Bridges multiple subnets to provide end-to-end internet connectivity between nodes
- Provides global addressing

Works across different link technologies

Different for each Internet “hop”
Layer 4: Transport Layer

- **Application**
- **Transport**
- **(Inter)Network**
- **Link**
- **Physical**

*End-to-end* communication between processes

Different services provided:
- TCP = *reliable byte stream*
- UDP = unreliable *datagrams*
Layer 7: Application Layer

Communication of whatever you wish

Can use whatever transport(s) is convenient

Freely structured

E.g.:
- Skype
- SMTP (email)
- HTTP (Web)
- Halo
- BitTorrent
Internet Layering ("Protocol Stack")

Implemented only at hosts, not at interior routers ("dumb network")
Internet Layering ("Protocol Stack")

Implemented everywhere
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D

Different Physical & Link Layers (Layers 1 & 2)

E.g., Wi-Fi

E.g., Ethernet
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D

Host A

Router 1

Host C

Router 2

Host D

Router 3

Host E

Router 4

Router 5

Router 6

Router 7

Same Network / Transport / Application Layers (3/4/7) (Routers **ignore** Transport & Application layers)

E.g., HTTP over TCP over IP
Key Concept #3: Protocols

• A protocol is an agreement on how to communicate

• Includes syntax and semantics
  – How a communication is specified & structured
    o Format, order messages are sent and received
  – What a communication means
    o Actions taken when transmitting, receiving, or timer expires

• E.g.: asking a question in lecture?
  1. Raise your hand.
  2. Wait to be called on.
  3. Or: wait for speaker to pause and vocalize
  4. If unrecognized (after timeout): vocalize w/ “excuse me”
**Example: IP Packet Header**

(Network layer / layer 3)

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Length</th>
<th>8-bit Type of Service (TOS)</th>
<th>16-bit Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16-bit Identification</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-bit Protocol</td>
<td>16-bit Header Checksum</td>
</tr>
<tr>
<td>8-bit Time to Live (TTL)</td>
<td>8-bit Protocol</td>
<td>3-bit Flags</td>
<td>13-bit Fragment Offset</td>
</tr>
<tr>
<td>32-bit Source IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit Destination IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Payload

**IP = Internet Protocol**
IP: “Best Effort” Packet Delivery

- Routers inspect destination address, locate “next hop” in forwarding table
  - Address = ~unique identifier/locator for the receiving host
  - (decrements TTL “Time To Live” field, drops packet if = 0)

- Only provides a “I’ll give it a try” delivery service:
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order
“Best Effort” is Lame! What to do?

• It’s the job of our Transport (layer 4) protocols to build services our apps need out of IP’s modest layer-3 service

• #1 workhorse: TCP (Transmission Control Protocol)

• TCP service:
  – Connection oriented (explicit set-up / tear-down)
    o End hosts (processes) can have multiple concurrent long-lived dialog
  – Reliable, in-order, byte-stream delivery
    o Robust detection & retransmission of lost data
  – Congestion control
    o Dynamic adaptation to network path’s capacity
    o (Also adaptation to receiver’s ability to absorb data)
TCP “Stream of Bytes” Service

Host A

Host B

Hosts don’t ever see packet boundaries, lost or corrupted packets, retransmissions, etc.
“Best Effort” is Lame! What to do?

• It’s the job of our Transport (layer 4) protocols to build services our apps need out of IP’s modest layer-3 service

• #1 workhorse: TCP (Transmission Control Protocol)

• TCP service:
  – Connection oriented (explicit set-up / tear-down)
    o End hosts (processes) can have multiple concurrent long-lived dialog
  – Reliable, in-order, byte-stream delivery
    o Robust detection & retransmission of lost data
  – Congestion control
    o Dynamic adaptation to network path’s capacity
    o (Also adaptation to receiver’s ability to absorb data)
### TCP Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td></td>
</tr>
<tr>
<td>Destination port</td>
<td></td>
</tr>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td></td>
</tr>
<tr>
<td>Advertised window</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>Urgent pointer</td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP Header

Ports are associated with OS processes

IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

Some port numbers are “well known” / reserved e.g. port 80 = HTTP

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>Flags</td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
</tbody>
</table>

Data
## TCP Header

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Sequence number**: Starting sequence number (byte offset) of data carried in this packet.

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Flags</th>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
TCP Header

<table>
<thead>
<tr>
<th></th>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>0</td>
<td>Flags</td>
</tr>
<tr>
<td>Advertised window</td>
<td></td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Acknowledgment gives seq # **just beyond** highest seq. received **in order**.

If sender sends **N** in-order bytes starting at seq **S** then ack for it will be **S+N**.
TCP Header

Uses include:

- acknowledging data ("ACK")
- setting up ("SYN")
- and closing connections ("FIN" and "RST")
Establishing a TCP Connection

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (**SYN+ACK**)
  - Host A sends an **ACK** to acknowledge the SYN+ACK

Each host tells its *Initial Sequence Number* (ISN) to the other host.
Spec says to pick based on local clock
Timing Diagram: 3-Way Handshaking

Client (initiator)

Active
Open

connect()

Server

Passive
Open

listen()

accept()

SYN, SeqNum = x

SYN + ACK, SeqNum = y, Ack = x + 1

ACK, Ack = y + 1
Host Names vs. IP addresses

• Host names
  – Examples: www.cnn.com and bbc.co.uk
  – Mnemonic name appreciated by humans
  – Variable length, full alphabet of characters
  – Provide little (if any) information about location

• IP addresses
  – Examples: 64.236.16.20 and 212.58.224.131
  – Numerical address appreciated by routers
  – Fixed length, binary number
  – Hierarchical, related to host location
Mapping Names to Addresses

• Domain Name System (DNS)
  – Hierarchical name space divided into zones
  – Zones distributed over collection of DNS servers
  – (Also separately maps addresses to names)

• Hierarchy of DNS servers
  – Root (hardwired into other servers)
  – Top-level domain (TLD) servers
  – “Authoritative” DNS servers (e.g. for berkeley.edu)

• Performing the translations
  – Each computer configured to contact a resolver
Distributed Hierarchical Database

Top-Level Domains (TLDs)

- com
- edu
- org
- ac
- uk
- zw
- arpa

Generic Domains
- bar
  - west
  - foo
- east
  - my

Country Domains
- ac
  - cam
  - usr
  - in-addr

Example Domain Names:
- my.east.bar.edu
- usr.cam.ac.uk
Example

Host at \texttt{xyz.poly.edu} wants IP address for \texttt{gaia.cs.umass.edu}

- **Requesting host**: \texttt{xyz.poly.edu}
- **Local DNS server** (resolver): \texttt{dns.poly.edu}
- **Root DNS server** (\texttt{.})
- **TLD DNS server** (\texttt{.edu})
- **Authoritative DNS server** (\texttt{umass.edu}, \texttt{cs.umass.edu})

\texttt{gaia.cs.umass.edu}
DNS Protocol

DNS protocol: *query* and *reply* messages, both with same message format

(Mainly uses UDP transport rather than TCP)

Message header:

- **Identification**: 16 bit # for query, reply to query uses same #
- Replies can include “Authority” (name server responsible for answer) and “Additional” (info client is likely to look up soon anyway)
- Replies have a Time To Live (in seconds) for **caching**
Bootstrapping Problem

• New host doesn’t have an IP address yet
  – So, host doesn’t know what source address to use

• Host doesn’t know who to ask for an IP address
  – So, host doesn’t know what destination address to use

• Solution: shout to “discover” server that can help
  – Broadcast a server-discovery message (layer 2)
  – Server(s) sends a reply offering an address
Dynamic Host Configuration Protocol

new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK

DHCP server

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)
Questions?