CMPS122: COMPUTER SECURITY

NETWORKING ATTACKS
NOTICES

» Lab #2 extended to Feb. 17 @ 23:59

» HW #3 due tonight
LAST TIME

- TCP/IP networking stack
  - Physical layer
  - Data link layer
  - Network layer
  - Transport layer
  - Application layer

- Concepts
  - *Protocols*: an agreement on how to communicate
  - *“Dumb” network*: Interior nodes have little knowledge of ongoing connections
  - *Layering*: Layers depend on services from lower layer, and provide services to upper layer
TODAY

- Network attacks (Link, IP, and TCP layers)
- Wireshark demo
General Communication Security Goals: CIA

• **Confidentiality:**
  – No one can *read* our data / communication unless we want them to

• **Integrity**
  – No one can *manipulate* our data / processing / communication unless we want them to

• **Availability**
  – We can *access* our data / conduct our processing / use our communication capabilities when we want to

• Also: no *additional* traffic other than ours …
Link-layer threats

- Confidentiality: eavesdropping (aka sniffing)
- Integrity: injection of spoofed packets
- Injection: delete legit packets (e.g., jamming)
Layers 1 & 2: General Threats?

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

Encoding bits to send them over a single physical link e.g. patterns of voltage levels / photon intensities / RF modulation
Eavesdropping

• For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), eavesdropping comes for “free”
  – Each attached system’s NIC (= Network Interface Card) can capture any communication on the subnet
  – Some handy tools for doing so
    o tcpdump / windump (low-level ASCII printout)
    o Wireshark (GUI for displaying 800+ protocols)
TCPDUMP: Packet Capture & ASCII Dumper

demo 2 % tcpdump -r all.trace2
reading from file all.trace2, link-type EN10MB (Ethernet)
21:39:37.772565 IP 10.0.1.9.62137 > all-systems.mcast.net.canon-bjnp2: UDP, length 16
523449627, win 65535, options [mss 1460,nop,wscale 3,nop,nop,TS val 429017455 ecr 0,sack
OK,eol], length 0
3585654832, ack 2523449628, win 14480, options [mss 1460,sackOK,TS val 1765826995 ecr 42
9017455,nop,wscale 9], length 0
, win 65535, options [nop,nop,TS val 429017456 ecr 1765826995], length 0
1:525, ack 1, win 65535, options [nop,nop,TS val 429017456 ecr 1765826995], length 524
25, win 31, options [nop,nop,TS val 1765827012 ecr 429017456], length 0
1:535, ack 525, win 31, options [nop,nop,TS val 1765827083 ecr 429017456], length 534
35, win 65535, options [nop,nop,TS val 429017457 ecr 1765827083], length 0
21:39:44.838031 IP 10.0.1.9.54277 > 10.0.1.255.canon-bjnp2: UDP, length 16
21:39:44.838213 IP 10.0.1.9.62896 > all-systems.mcast.net.canon-bjnp2: UDP, length 16
Wireshark: GUI for Packet Capture/Exam.

Frame 10: 600 bytes on wire (4800 bits), 600 bytes captured (4800 bits)
EtherType 806: Ethernet II, Src: Apple_A:fa:41 [00:25:00:ee:fa:41], Dst: Apple_A:41:eb:00 [e4:ce:8f:41:eb:00]
Transmission Control Protocol, Src Port: http (80), Dst Port: 61901 (61901), Seq: 1, Ack: 525, Len: 534

Hypertext Transfer Protocol
Wireshark: GUI for Packet Capture/Exam.
Wireshark: GUI for Packet Capture/Exam.
Stealing Photons

1. Micro-bend clamping device
2. Optical photo detector
3. Optical-electrical converter
4. Laptop

Jacket
Cladding
Active fiber optic cable
Lost light <1%

Image of a physical device and a ruler in the background.
Operation Ivy Bells

By Matthew Carle
Military.com

At the beginning of the 1970's, divers from the specially-equipped submarine, USS Halibut (SSN 587), left their decompression chamber to start a bold and dangerous mission, code named "Ivy Bells".

The Regulus guided missile submarine, USS Halibut (SSN 587) which carried out Operation Ivy Bells.

In an effort to alter the balance of Cold War, these men scoured the ocean floor for a five-inch diameter cable carry secret Soviet communications between military bases.

The divers found the cable and installed a 20-foot long listening device on the cable, designed to attach to the cable without piercing the casing, the device recorded all communications that occurred. If the cable malfunctioned and the Soviets raised it for repair, the bug, by design, would fall to the bottom of the ocean. Each month Navy divers retrieved the recordings and installed a new set of tapes.

Upon their return to the United States, intelligence agents from the NSA analyzed the recordings and tried to decipher any encrypted information. The Soviets apparently were confident in the security of their communications lines, as a surprising amount of sensitive information traveled through the lines without encryption.

A bug similar to this one is now on exhibit at the KGB museum in Moscow.
Link-Layer Threat: Disruption

• If attacker sees a packet he doesn’t like, he can jam it (integrity)

• Attacker can also overwhelm link-layer signaling, e.g., jam WiFi’s RF (denial-of-service)
Link-Layer Threat: Disruption

• If attacker sees a packet he doesn’t like, he can jam it (integrity)

• Attacker can also overwhelm link-layer signaling, e.g., jam WiFi’s RF (denial-of-service)

• There’s also the heavy-handed approach …
**Sabotage attacks knock out phone service**

Nanette Asimov, Ryan Kim, Kevin Fagan, Chronicle Staff Writers

Friday, April 10, 2009

(04-10) 04:00 PDT SAN JOSE --

Police are hunting for vandals who chopped fiber-optic cables and killed landlines, cell phones and Internet service for tens of thousands of people in Santa Clara, Santa Cruz and San Benito counties on Thursday.

The sabotage essentially froze operations in parts of the three counties at hospitals, stores, banks and police and fire departments that rely on 911 calls, computerized medical records, ATMs and credit and debit cards.

The full extent of the havoc might not be known for days, emergency officials said as they finished repairing the damage late Thursday.

Whatever the final toll, one thing is certain: Whoever did this is in a world of trouble if he, she or they get caught.

"I pity the individuals who have done this," said San Jose Police Chief Rob Davis.

Ten fiber-optic cables carrying were cut at four locations in the predawn darkness. Residential and business customers quickly found that telephone service was perhaps more laced into their everyday needs than they thought. Suddenly they couldn't draw out money, send text messages, check e-mail or Web sites, call anyone for help, or even check on friends or relatives down the road.

Several people had to be driven to hospitals because they were unable to summon ambulances. Many businesses lapsed into idleness for hours, without the ability to contact associates or customers.

More than 50,000 landline customers lost service - some were residential, others were business lines that needed the connections for ATMs, Internet and bank card transactions. One line alone could affect hundreds of users.
Link-Layer Threat: Spoofing

• Attacker can inject spoofed packets, and lie about the source address

| D | C | Hello world! |
Physical/Link-Layer Threats: Spoofing

- With physical access to a local network, attacker can create any message they like
  - When with a bogus source address: *spoofing*

- When using a typical computer, may require root/administrator to have full freedom

- Particularly powerful when combined with *eavesdropping*
  - Because attacker can understand exact state of victim’s communication and craft their spoofed traffic to match it
- Spoofing w/o eavesdropping = *blind spoofing*
On-path vs Off-path Spoofing

Host A communicates with Host D

On-path

Off-path
Spoofing on the Internet

• On-path attackers can see victim’s traffic ⇒ spoofing is easy

• Off-path attackers can’t see victim’s traffic
  – They have to resort to blind spoofing
  – Often must guess/infer header values to succeed
    o We then care about work factor: how hard is this
  – But sometimes they can just brute force
    o E.g., 16-bit value: just try all 65,536 possibilities!

• When we say an attacker “can spoof”, we usually mean “w/ reasonable chance of success”
Layer 3: General Threats?

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
</tr>
<tr>
<td>4</td>
<td>Transport</td>
</tr>
<tr>
<td>3</td>
<td>(Inter)Network</td>
</tr>
<tr>
<td>2</td>
<td>Link</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
</tr>
</tbody>
</table>

Bridges multiple “subnets” to provide end-to-end internet connectivity between nodes

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

Payload

IP = Internet Protocol
IP-Layer Threats

• Can set arbitrary source address
  – “Spoofing” - receiver has no idea who you are
  – Could be blind, or could be coupled w/ sniffing
  – Note: many attacks require two-way communication
    o So successful off-path/blind spoofing might not suffice

• Can set arbitrary destination address
  – Enables “scanning” – brute force searching for hosts

• Can send like crazy (flooding)
  – IP has no general mechanism for tracking overuse
  – IP has no general mechanism for tracking consent
  – Very hard to tell where a spoofed flood comes from!

• If attacker can manipulate routing, can bring traffic to themselves for eavesdropping (not easy)
LAN Bootstrapping: DHCP

• 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

A new client sends a DHCP discover message (broadcast) to request an IP address.

A DHCP offer message includes the IP address, DNS server, gateway router, and how long the client can have these (lease time).

The client sends a DHCP request message (broadcast) to accept the offer.

The DHCP server responds with a DHCP ACK message.
Dynamic Host Configuration Protocol

- New client
- DHCP discover (broadcast)
- DHCP offer
- DHCP request (broadcast)
- DHCP ACK

Threats?

“Offer” message includes IP address, DNS server, “gateway router,” and how long client can have these (“lease” time)
Dynamic Host Configuration Protocol

DHCP discover (broadcast)

DHCP offer

Attacker on same subnet can hear new host’s DHCP request

DHCP request (broadcast)

DHCP ACK

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)
Dynamic Host Configuration Protocol

A new client sends a DHCP discover (broadcast) message.

DHCP server responds with a DHCP offer message, which includes an IP address, DNS server, gateway router, and how long the client can have these (lease time).

Attacker can race the actual server; if they win, replace DNS server and/or gateway router.
DHCP Threats

• Substitute a fake DNS server
  – Redirect any of a host’s lookups to a machine of attacker’s choice

• Substitute a fake gateway router
  – Intercept all of a host’s off-subnet traffic
    o (even if not preceded by a DNS lookup)
  – Relay contents back and forth between host and remote server and modify however attacker chooses

• An invisible Man In The Middle (MITM)
  – Victim host has no way of knowing it’s happening
    o (Can’t necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)

• How can we fix this? **Hard**
TCP

These plus IP addresses define a given connection

<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>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
</tbody>
</table>

Data
TCP

Defines where this packet fits within the sender’s bytestream

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

<table>
<thead>
<tr>
<th>Sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
TCP Conn. Setup & Data Exchange

**Client (initiator)**
IP address 1.2.1.2, port 3344

**Server**
IP address 9.8.7.6, port 80

- **SrcA=1.2.1.2, SrcP=3344,**
  **DstA=9.8.7.6, DstP=80, SYN, Seq = x**

- **SrcA=9.8.7.6, SrcP=80,**
  **DstA=1.2.1.2, DstP=3344, SYN+ACK, Seq = y, Ack = x+1**

- **SrcA=1.2.1.2, SrcP=3344,**
  **DstA=9.8.7.6, DstP=80, ACK, Seq = x+1, Ack = y+1**

- **SrcA=1.2.1.2, SrcP=3344,**
  **DstA=9.8.7.6, DstP=80, ACK, Seq=x+1, Ack = y+1, Data=“GET /login.html”**

- **SrcA=9.8.7.6, SrcP=80,**
  **DstA=1.2.1.2, DstP=3344,**
  **ACK, Seq = y+1, Ack = x+16, Data=“200 OK … <html> …”**
TCP Threat: Data Injection

- If attacker knows ports & sequence numbers (e.g., on-path attacker), attacker can inject data into any TCP connection
  - Receiver B is none the wiser!

- Termed TCP connection hijacking (or “session hijacking”)
  - A general means to take over an already-established connection!

- We are toast if an attacker can see our TCP traffic!
  - Because then they immediately know the port & sequence numbers
TCP Data Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker
IP address 6.6.6.6, port N/A

Client dutifully processes as server’s response

\[
\text{SrcA}=1.2.1.2, \text{SrcP}=3344, \text{DstA}=9.8.7.6, \text{DstP}=80, \\
\text{ACK, Seq}=x+1, \text{Ack}=y+1, \text{Data}="\text{GET /login.html}"
\]

\[
\text{SrcA}=9.8.7.6, \text{SrcP}=80, \\
\text{DstA}=1.2.1.2, \text{DstP}=3344, \\
\text{ACK, Seq} = y+1, \text{Ack} = x+16 \\
\text{Data}="\text{200 OK ... <poison> ...}"
\]
TCP Data Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker
IP address 6.6.6.6, port N/A

Client ignores since already processed that part of bytestream
TCP Threat: Disruption

• Is it possible for an on-path attacker to shut down a TCP connection if they can see our traffic?
  • YES: they can infer the port and sequence numbers – they can insert fake data, too! (Great Firewall of China)
TCP Threat: Blind Hijacking

• Is it possible for an off-path attacker to inject into a TCP connection even if they can’t see our traffic?
  • YES: if somehow they can infer or guess the port and sequence numbers
TCP Threat: Blind Spoofing

• Is it possible for an off-path attacker to create a fake TCP connection, even if they can’t see responses?

• **YES:** if somehow they can infer or guess the TCP initial sequence numbers

• Why would an attacker want to do this?
  – Perhaps to leverage a server’s trust of a given client as identified by its IP address
  – Perhaps to frame a given client so the attacker’s actions during the connections can’t be traced back to the attacker
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port N/A

Server
IP address 9.8.7.6, port 80

Blind Attacker

Attacker’s goal:

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = z

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = z+1

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = z+1, ACK = y+1

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = z+1, ACK = y+1, Data = “GET /transfer-money.html”
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port NA

Server
IP address 9.8.7.6, port 80

Blind Attacker

SrcA=1.2.1.2, SrcP=5566,
DstA=9.8.7.6, DstP=80, SYN, Seq = z

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = x+1

Small Note #1: if alleged client receives this, will be confused ⇒ send a RST back to server …
… So attacker may need to hurry!
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port NA

Blind Attacker
SrcA=1.2.1.2, SrcP=5566,
DstA=9.8.7.6, DstP=80, SYN, Seq = z

Server
IP address 9.8.7.6, port 80

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = z+1

Big Note #2: attacker doesn’t get to see this packet!
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port N/A

Server
IP address 9.8.7.6, port 80

Blind Attacker

SrcA=1.2.1.2, SrcP=5566,
DstA=9.8.7.6, DstP=80, SYN, Seq = z

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = z+1

So how can the attacker figure out what value of y to use for their ACK?

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6,
DstP=80, ACK, Seq = z+1, ACK = y+1

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6,
DstP=80, ACK, Seq = z+1, ACK = y+1,
Data = “GET /transfer-money.html”
Reminder: Establishing a TCP Connection

Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

Hmm, any way for the attacker to know *this*?

How Do We Fix This?

Use a (Pseudo)-Random ISN

Sure – make a non-spoofed connection *first*, and see what server used for ISN *y* then!
Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully **terminate** by forging a RST packet
  - **Inject** *(spoof)* data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - *Remains a major threat today*
Summary of TCP Security Issues

• An attacker who can observe your TCP connection can manipulate it:
  – Forcefully **terminate** by forging a RST packet
  – **Inject** (*spoof*) data into either direction by forging data packets
  – Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  – *Remains a major threat today*

• If attacker could **predict** the ISN chosen by a server, could “blind spoof” a connection to the server
  – Makes it appear that host ABC has connected, and has sent data of the attacker’s choosing, when in fact it hasn’t
  – *Undermines any security based on trusting ABC’s IP address*
  – Allows attacker to “frame” ABC or otherwise *avoid detection*
  – **Fixed** (mostly) today by choosing **random** ISNs
Summary of IP security

• No security against on-path attackers
  – Can sniff, inject packets, mount TCP spoofing, TCP hijacking, man-in-the-middle attacks
  – Typical example: wireless networks, malicious network operator

• Reasonable security against off-path attackers
  – TCP is basically secure, but UDP and IP are not
Extra Material
Sequence Numbers

Host A

ISN (initial sequence number)

Sequence number = 1\textsuperscript{st} byte

TCP Data

TCP HDR

ACK sequence number = next expected byte

Host B
TCP Threat: Disruption

• Normally, TCP finishes (“closes”) a connection by each side sending a FIN control message
  – Reliably delivered, since other side must ack

• But: if a TCP endpoint finds unable to continue (process dies; info from other “peer” is inconsistent), it abruptly terminates by sending a RST control message
  – Unilateral
  – Takes effect immediately (no ack needed)
  – Only accepted by peer if has correct* sequence number
<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<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>0</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
</tbody>
</table>

Data
Abrupt Termination

- A sends a TCP packet with RESET (RST) flag to B
  - E.g., because app. process on A crashed
  - (Could instead be that B sends a RST to A)

- Assuming that the sequence numbers in the RST fit with what B expects, That’s It:
  - B’s user-level process receives: ECONNRESET
  - No further communication on connection is possible
TCP Threat: Disruption

• Normally, TCP finishes ("closes") a connection by each side sending a FIN control message
  – Reliably delivered, since other side must ack

• But: if a TCP endpoint finds unable to continue (process dies; info from other "peer" is inconsistent), it abruptly terminates by sending a RST control message
  – Unilateral
  – Takes effect immediately (no ack needed)
  – Only accepted by peer if has correct* sequence number

• So: if attacker knows ports & sequence numbers, can disrupt any TCP connection
TCP RST Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker
IP address 6.6.6.6, port N/A

\[ \text{SrcA}=1.2.1.2, \text{SrcP}=3344, \text{DstA}=9.8.7.6, \text{DstP}=80, \]
\[ \text{ACK, Seq}=x+1, \text{Ack} = y+1, \text{Data}=“\text{GET /login.html”} \]

\[ \text{SrcA}=9.8.7.6, \text{SrcP}=80, \]
\[ \text{DstA}=1.2.1.2, \text{DstP}=3344, \]
\[ \text{RST, Seq} = y+1, \text{Ack} = x+16 \]

Client dutifully removes connection
TCP RST Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker
IP address 6.6.6.6, port N/A

SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80,
ACK, Seq=x+1, Ack = y+1, Data="GET /login.html"

Client rejects since no active connection

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344,
RST, Seq = y+1, Ack = x+16

SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344,
ACK, Seq = y+1, Ack = x+16, Data="200 OK ... <html> ...

...
Threats to Comm. Security Goals

• Attacks can subvert each type of goal
  – Confidentiality: eavesdropping / theft of information
  – Integrity: altering data, manipulating execution (e.g., code injection)
  – Availability: denial-of-service

• Attackers can also combine different types of attacks towards an overarching goal
  – E.g. use eavesdropping (confidentiality) to construct a spoofing attack (integrity) that tells a server to drop an important connection (denial-of-service)
TCP’s Rate Management

Unless there’s loss, TCP doubles data in flight every “round-trip”. All TCPs expected to obey (“fairness”).

Mechanism: for each arriving ack for new data, increase allowed data by 1 maximum-sized packet

E.g., suppose maximum-sized packet = 100 bytes
Protocol Cheating

How can the destination (receiver) get data to come to them faster than normally allowed?

**ACK-Splitting**: each ack, even though partial, increases allowed data by one maximum-sized packet

How do we defend against this?

Change rule to require “full” ack for all data sent in a packet
Protocol Cheating

How can the destination (receiver) still get data to come to them faster than normally allowed?

*Opportunistic ack'ing*: acknowledge data not yet seen!

How do we defend against *this*?
Keeping Receivers Honest

• Approach #1: if you receive an ack for data you haven’t sent, kill the connection
  – Works only if receiver acks too far ahead

• Approach #2: follow the “round trip time” (RTT) and if ack arrives too quickly, kill the connection
  – Flaky: RTT can vary a lot, so you might kill innocent connections

• Approach #3: make the receiver prove they received the data
  – Add a nonce (“random” marker) & require receiver to include it in ack. Kill connections w/ incorrect nonces
  o (nonce could be function computed over payload, so sender doesn’t explicitly transmit, only implicitly)

Note: a protocol change