Lecture 4
Gaining Access using Network Attacks

ENGG5105/CSCI5470 Computer and Network Security

Spring 2014

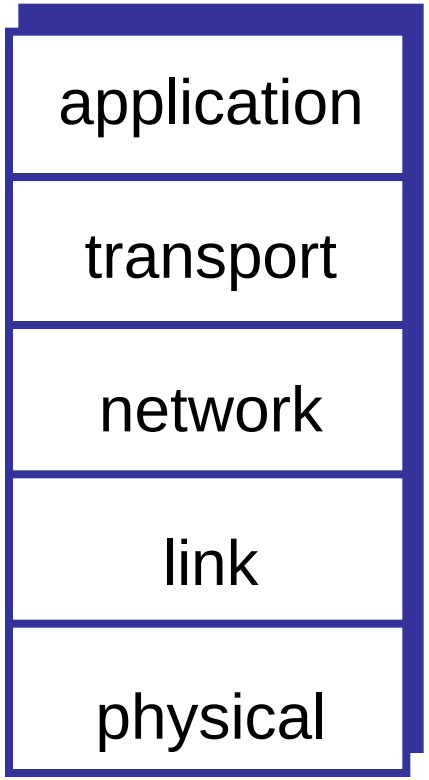
Patrick P. C. Lee

Layering

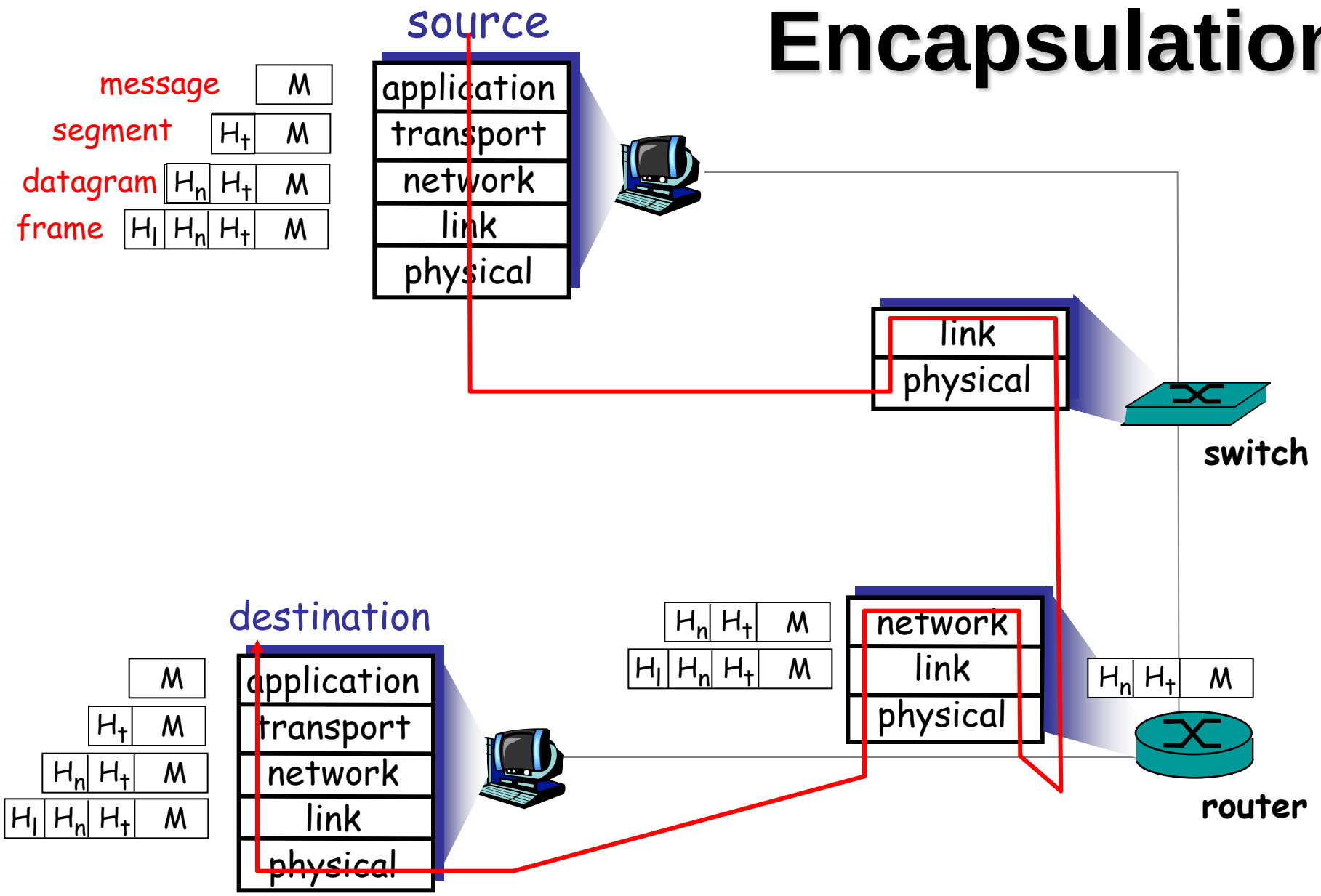
- The Internet is built on several protocols, each corresponds to a **layer**
- Layering allows modularization, which eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - different layers just agree on the interface to exchange messages, but implementation of each layer is independent of other layers'
- Encapsulation is the process that includes layers into application message

Internet protocol stack

- application: supporting network applications
  - FTP, SMTP, HTTP
- transport: process-process data transfer
  - TCP, UDP
- network: routing of datagrams from source to destination
  - IP, routing protocols
- link: data transfer between neighboring network elements
  - PPP, Ethernet
- physical: bits “on the wire”

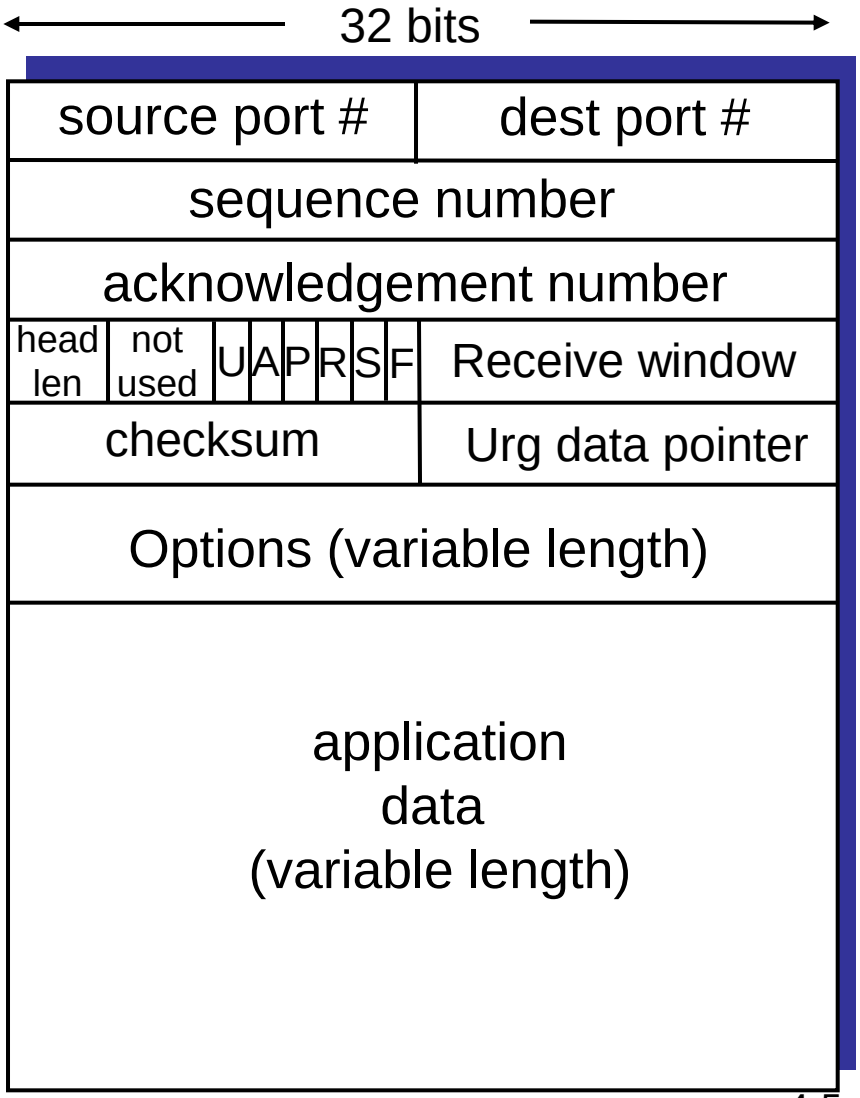


Encapsulation



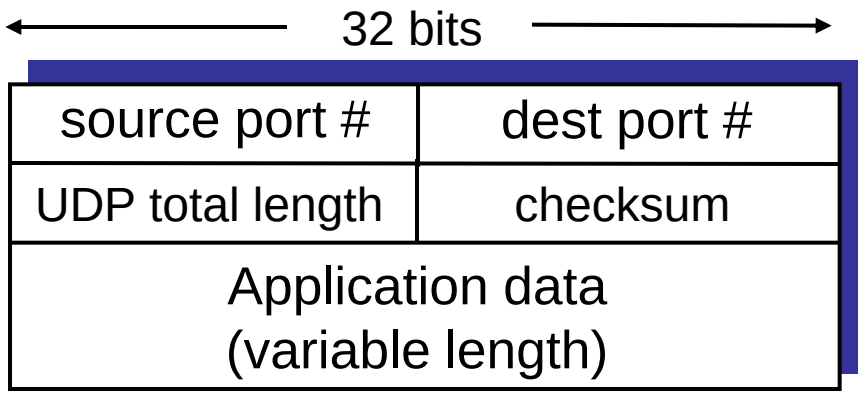
TCP Segment Format

- Port number
  identifies a process within a host
- TCP header size = (20 + length(options)) bytes



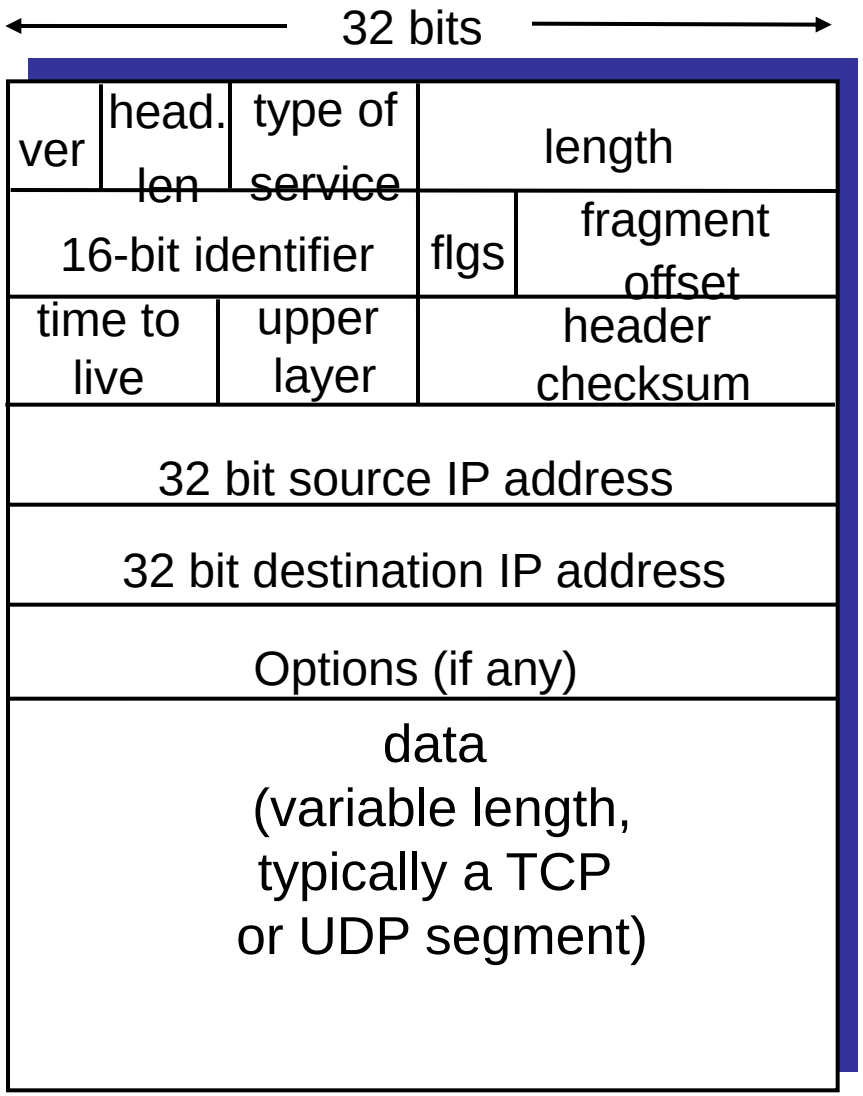
UDP Segment Format

- UDP header size
  = 8 bytes



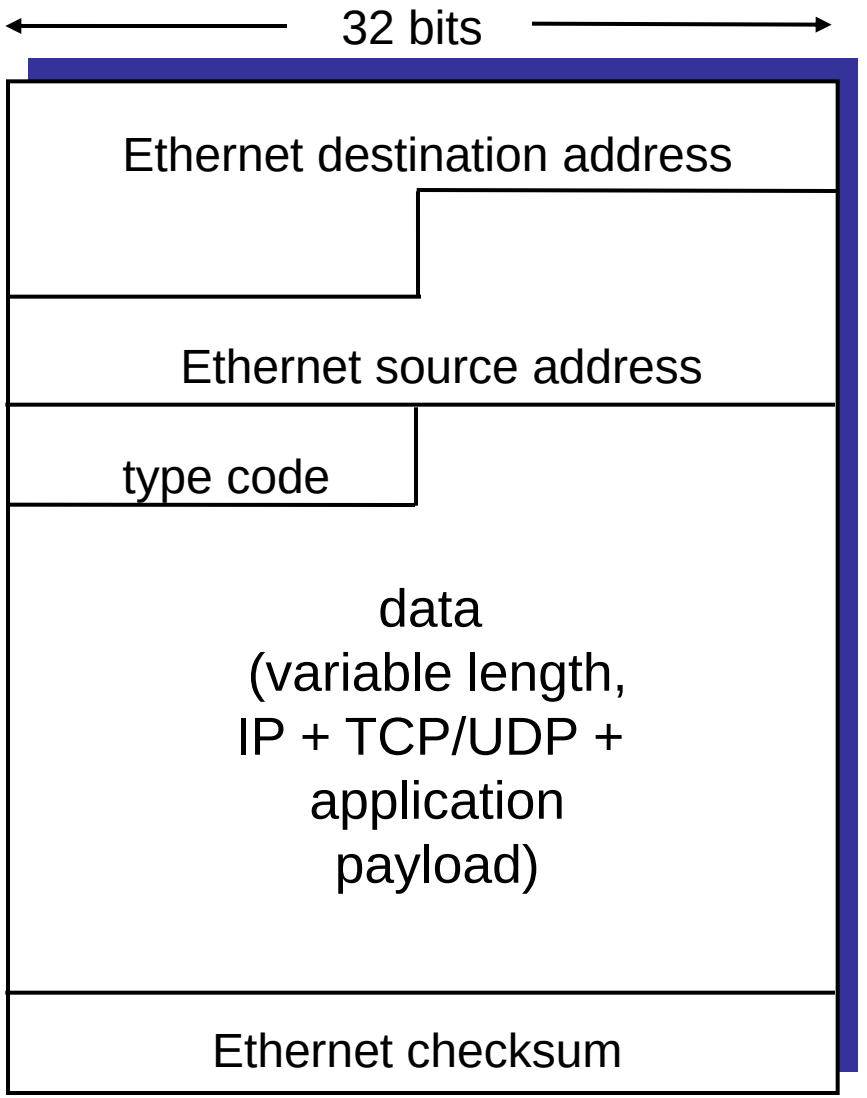
IP Datagram Format

- IP address is a 32-bit identifier of a host and an interface
- IP header size = (20 + length(options)) bytes



Ethernet Frame Format

- Ethernet header size = 14 bytes



Threat Model

- Our goal is to exploit the fundamental features of network protocols to gain malicious access
- Network attacks considered here will work for any medium (wired/wireless, switches), in LAN or the Internet
- Attacks that we consider:
  - Sniffing: steals your information
  - TCP exploits: kills your connection
  - Session hijacking: controls your connection
  - Netcat: creates a backdoor that accepts access from everywhere

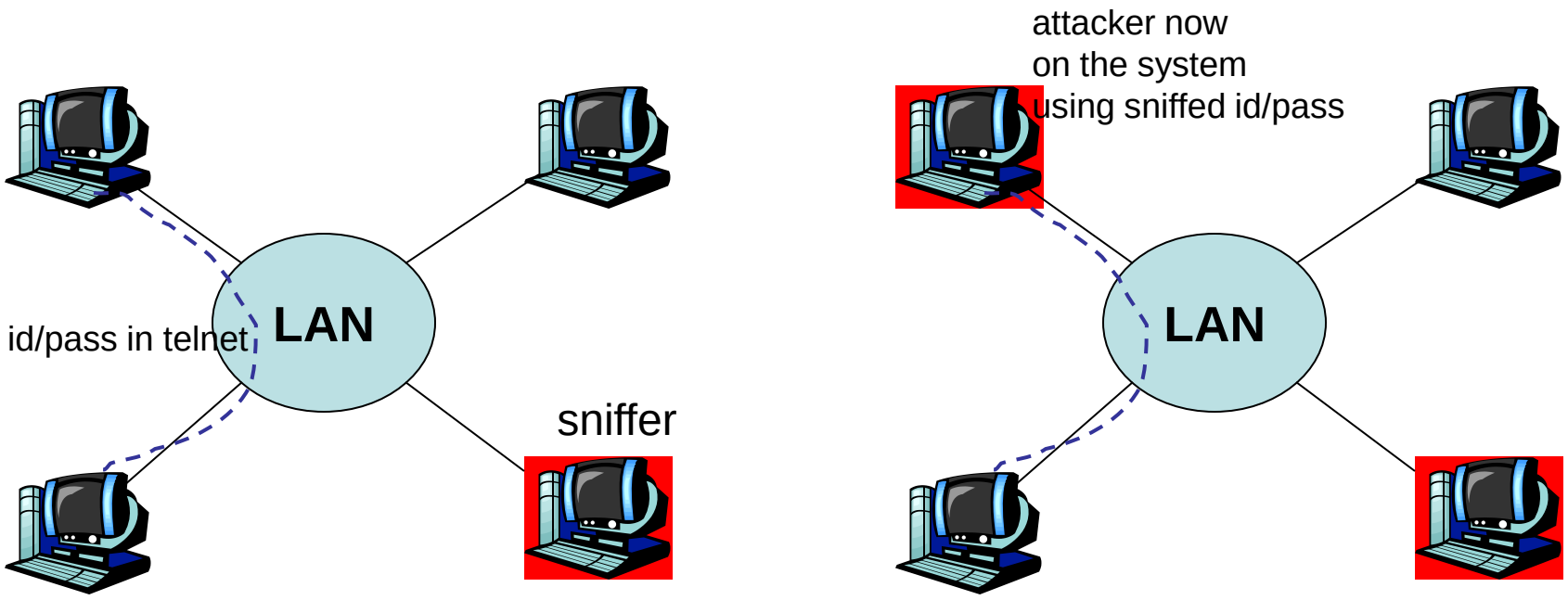
Roadmap

- Passive Sniffing
- Active Sniffing with ARP Spoofing
- TCP Exploits
- Session Hijacking
- Netcat

Sniffing

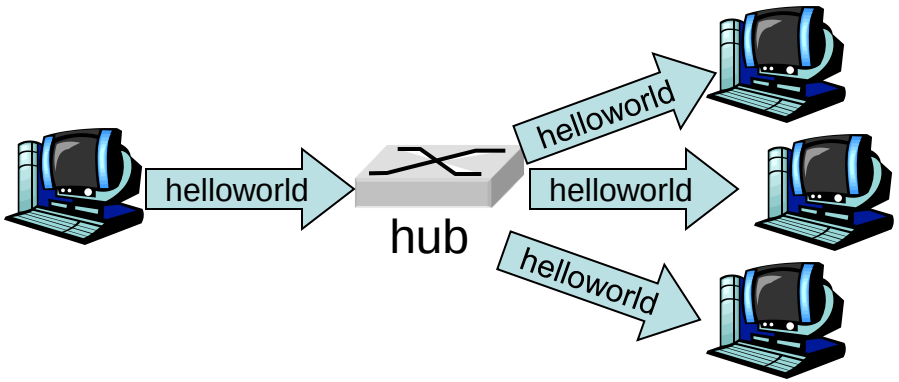
- Sniffing (or eavesdropping) refers to the activity that gathers traffic from the local network
  - Can see anything - all data from link layer to application layer (e.g., your password, if not encrypted)
  - Put interface in promiscuous mode, to listen all transmissions even destination address is not mine
- Used by attackers to steal sensitive information
- But sniffing is great in general, as it's used by administrators to monitor network behavior

Sniffing – Island Hopping Attack



- An attacker who took over one system sniffs for user IDs and passwords in other systems, and take over them one by one.

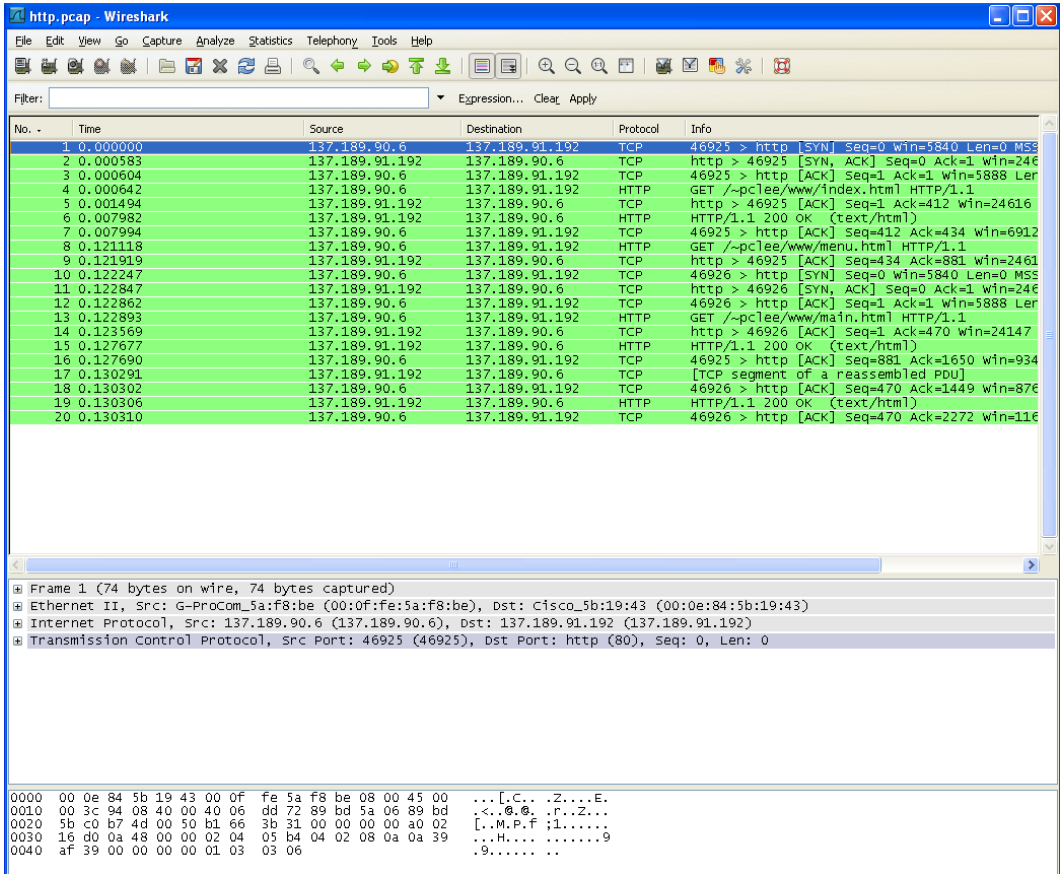
Passive Sniffing via a Hub



- **Hub** is a physical layer (dumb) repeater
  - bits coming in one link go out to all other links at same rate
  - if promiscuous mode enabled, bits can be seen
  - no frame buffering, no collision detection
  - slow
- Suitable for **passive sniffing**: not injecting new traffic
- Hub is rarely used today, but passive sniffing applies to any broadcast-type shared medium (e.g., wireless)

Passive Sniffer - Wireshark

- **Wireshark** – an open-source tool that provides a nice breakdown on every captured packet
  - http://www.wireshark.org
- Wireshark is built on libpcap library, same as tcpdump



Write your Own Sniffer

- Use libpcap to write your own sniffer
  - You can get the packet timestamp at which the packet is captured
  - You can get the full/partial packet payload
- Three steps:
  - pcap_open_live() – open the interface
  - pcap_next() – get one packet at a time
  - pcap_close() – close the interface

Write your Own Sniffer

- pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf);
  - device – the interface name (e.g,. eth0)
  - snaplen – the size of packet that you want to capture (smaller, allow faster capture)
  - promisc – 1 if device is put in promiscuous mode
  - to_ms – read timeout in milliseconds
    - typical value = 1000 to allow batch-copy of packets from kernel space to user space
  - errbuf – returned error message (if any)
- Return a pcap handle of type pcap_t
- You can use pcap_open_offline() to read from an already captured trace file.

Write your Own Sniffer

- const u_char *pcap_next(pcap_t *p,
  struct pcap_pkthdr *h);
  - Get one packet at a time. The pointer to the raw packet is returned
  - Packet timestamp is returned in pcap_pkthdr.
- pcap_close(pcap_t* p);
  - Close the pcap handle
- Use /usr/include/netinet/*.h to decode packet headers
- See **sample code**

Write your Own Sniffer

- How to compile?
  - gcc –Wall sniffer.c –o sniffer –lpcap
  - Make sure the libpcap library is installed
- You need the root privilege to enable the promiscuous mode of the network card
- See http://www.tcpdump.org/pcap.htm for more details

Roadmap

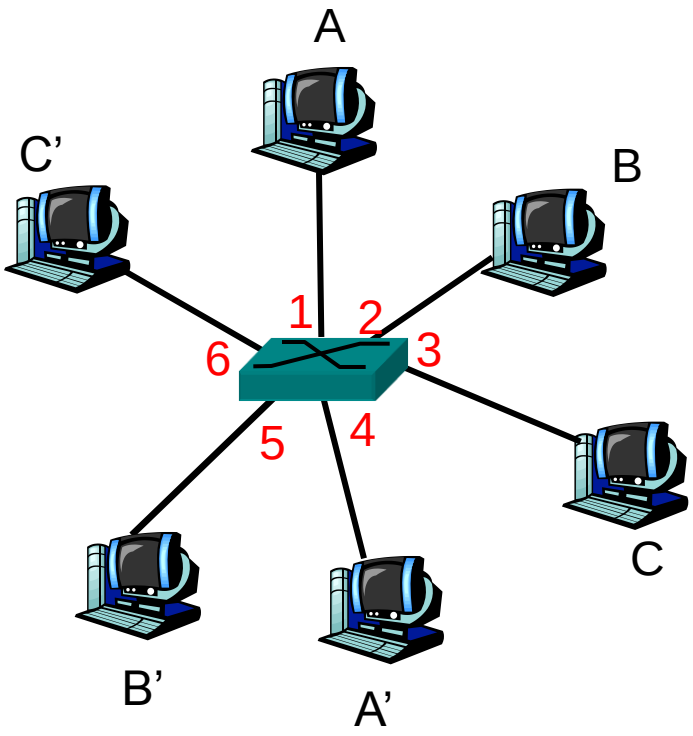
- Passive Sniffing
- Active Sniffing with ARP Spoofing
- TCP Exploits
- Session Hijacking
- Netcat

Switches

- **Switches** are now widely used to connect machines in a LAN
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links
  - self-learning mappings of ports and MAC addresses, plug and play
- Solve the passive sniffing problem, but not bullet-proof to sniffing… (details coming)

Switches

- *switching:* A-to-A' and B-to-B' simultaneously, without collisions
- Each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
- Switch learns entries from incoming frames



switch with six interfaces (1,2,3,4,5,6)

Active Sniffing

- Can we sniff through a switch?
- Yes, if you're allowed to insert traffic
- MAC address flooding
  - Sends packets with random spoofed MAC addresses
  - Switches address memory eventually exhausted, and revert to a hub-like mode
  - May not work for some switches if they limit the number of addresses that can be learned per port
- ARP Spoofing
  - Light-weight attack

ARP (Address Resolution Protocol)

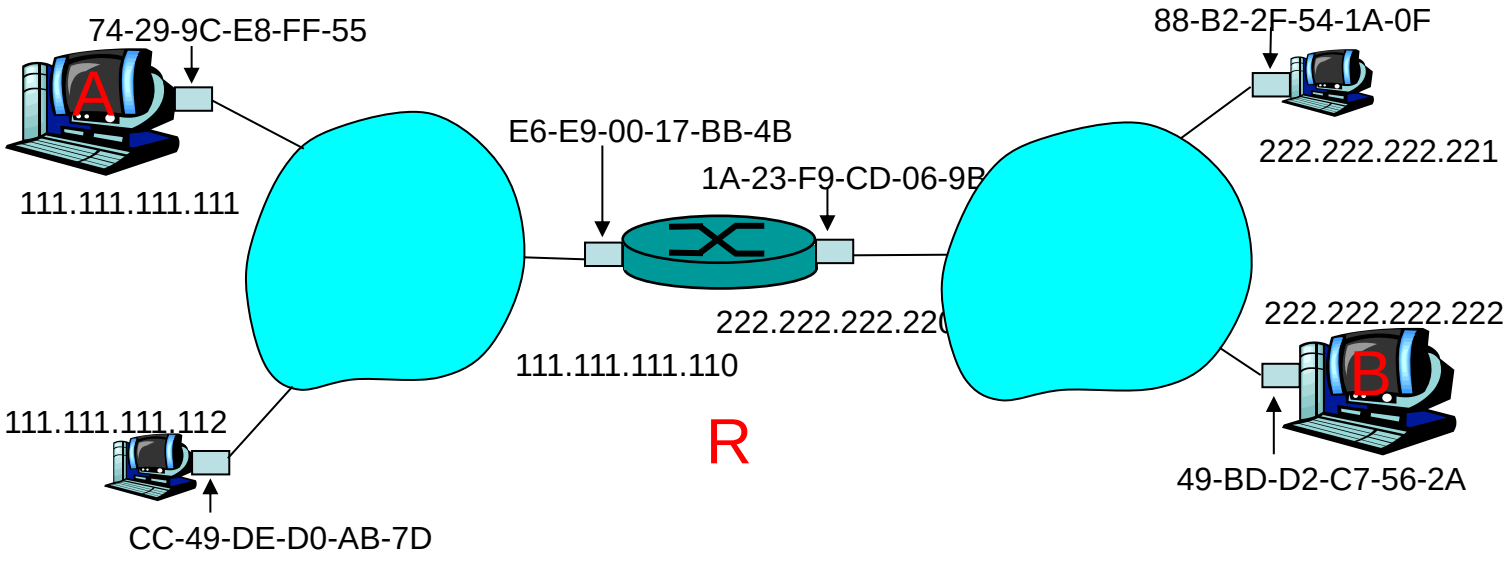
- A machine usually has two addresses
  - MAC address: layer-2 address for LAN communication
    - e.g., 00:0f:fe:5a:f8:b2
  - IP address: layer-3 address for WAN (e.g., Internet) communication
    - e.g., 137.189.90.6
- Question: how does host A tell the MAC address of host B with its IP address?
  - Answer: Address Resolution Protocol (ARP)

ARP (Address Resolution Protocol)

- Each IP node (host, router) on LAN has ARP table that stores IP/MAC address mappings
  - < IP address; MAC address; TTL>
  - TTL typically lasts for several minutes
- ARP is a simple request-response protocol
  - Node A broadcasts a ARP query, containing B's IP address
  - B replies to A with its MAC address
- See **arp.pcap**

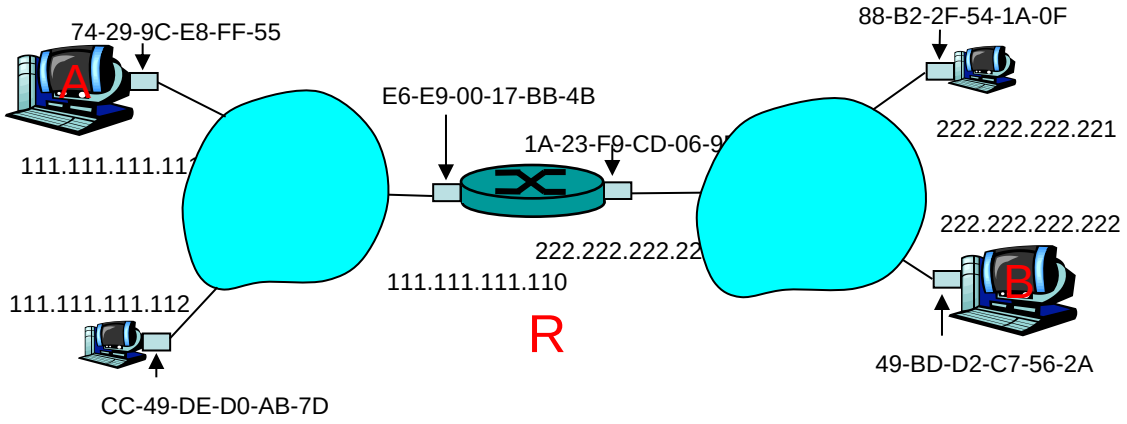
How ARP Helps Routing?

- Example of how ARP is used
- Suppose A wants to route packets to B



How ARP Helps Routing?

- A creates IP datagram with source A, destination B
- A uses ARP to get R's MAC address for 111.111.111.110
- A creates link-layer frame with R's MAC address as dest, frame contains A-to-B IP datagram
- A's NIC sends frame
- R's NIC receives frame
- R removes IP datagram from Ethernet frame, sees its destined to B
- R uses ARP to get B's MAC address
- R creates frame containing A-to-B IP datagram sends to B



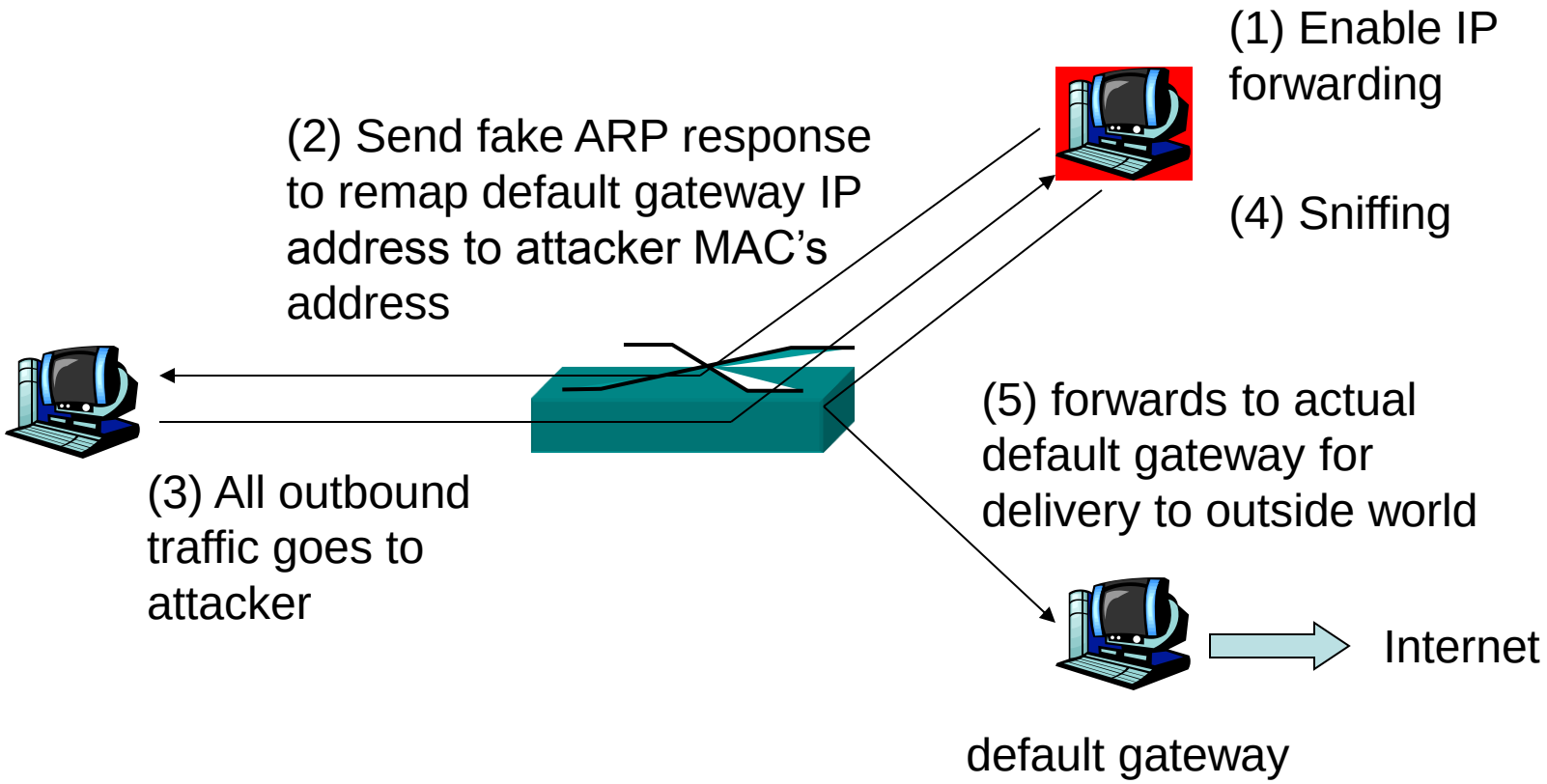
ARP Spoofing

- ARP spoofing aims to mess up the IP/MAC address mappings stored inside a host
- Idea:
  - Attacker sends a spoofed ARP request packet or a spoofed ARP reply packet
  - ARP is stateless. The victim will accept the ARP reply even if it doesn't issue an ARP request before
  - To evade detection, attacker enables IP forwarding and pretends to be default gateway. All sniffed traffic goes through itself first, and then to default gateway
    - In Linux, echo "1" > /proc/sys/net/ipv4/ip_forward

ARP Spoofing

- Gratuitous ARPs
  - Gratuitous ARP requests
    - source and destination IP set to the IP issuing the packets, and destination MAC address is set to broadcast. No reply packets will be sent.
  - Gratuitous ARP replies:
    - A reply to which no request issued before.
- Useful, e.g., when a new network card is installed, so gratuitous ARPs are announced for the new change.
- But abused by attackers.

ARP Spoofing



- ARP spoofing redirects traffic, allowing the attacker to sniff a switched LAN.

Example: ARP Spoofing

- Client (target victim)
  - IP address: 10.0.1.1
  - MAC address: 00:0c:29:62:96:db
- Default gateway
  - IP address: 10.0.1.30
  - MAC address: 00:0c:29:d4:2e:27
- Attacker
  - IP address: 10.0.1.2
  - MAC address: 00:0c:29:97:a8:4f
- Use the DSniff tool (http://monkey.org/~dugsong/dsniff)
  - Issue from attacker: **`arpspoof -t  10.0.1.1 10.0.1.30`**
- See **arpspoof.pcap**

DSniff

- DSniff is a collection of tools for network auditing and penetration testing. But attackers use it for malicious purposes.
  - you can do a "`yum install dsniff`" in Fedora
- Some DSniff functions/tools:
  - Password sniffing
  - ARP spoofing (arpspoof)
  - Killing active TCP connections (tcpkill)
  - Monkey-in-the-middle attacks (sshmitm, webmitm)

Sniffing Defenses

- Goal is to prevent information from being stolen
  - Use secure protocols
    - https, ssh, sftp, scp
  - Use switches over hubs
  - Configure switch ports of specific MAC addresses
  - Static ARP tables on end machines

Roadmap

- Passive Sniffing
- Active Sniffing with ARP Spoofing
- TCP Exploits
- Session Hijacking
- Netcat

Packet Spoofing

- How to craft my own ARP packets?
- Use socket programming. Create a socket of type SOCK_PACKET
  - fd = socket(AF_INET, SOCK_PACKET, htons(ETH_P_ALL));
- Craft your own headers/payload from the link layer to the application layer.

Packet Spoofing

- Similar idea for IP address spoofing
  - E.g., send DNS queries with spoofed source IP address, so that all DNS replies are all redirected to the spoofed victim address.
- Create a raw socket
  - fd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
  - You can craft everything from IP header to application payload

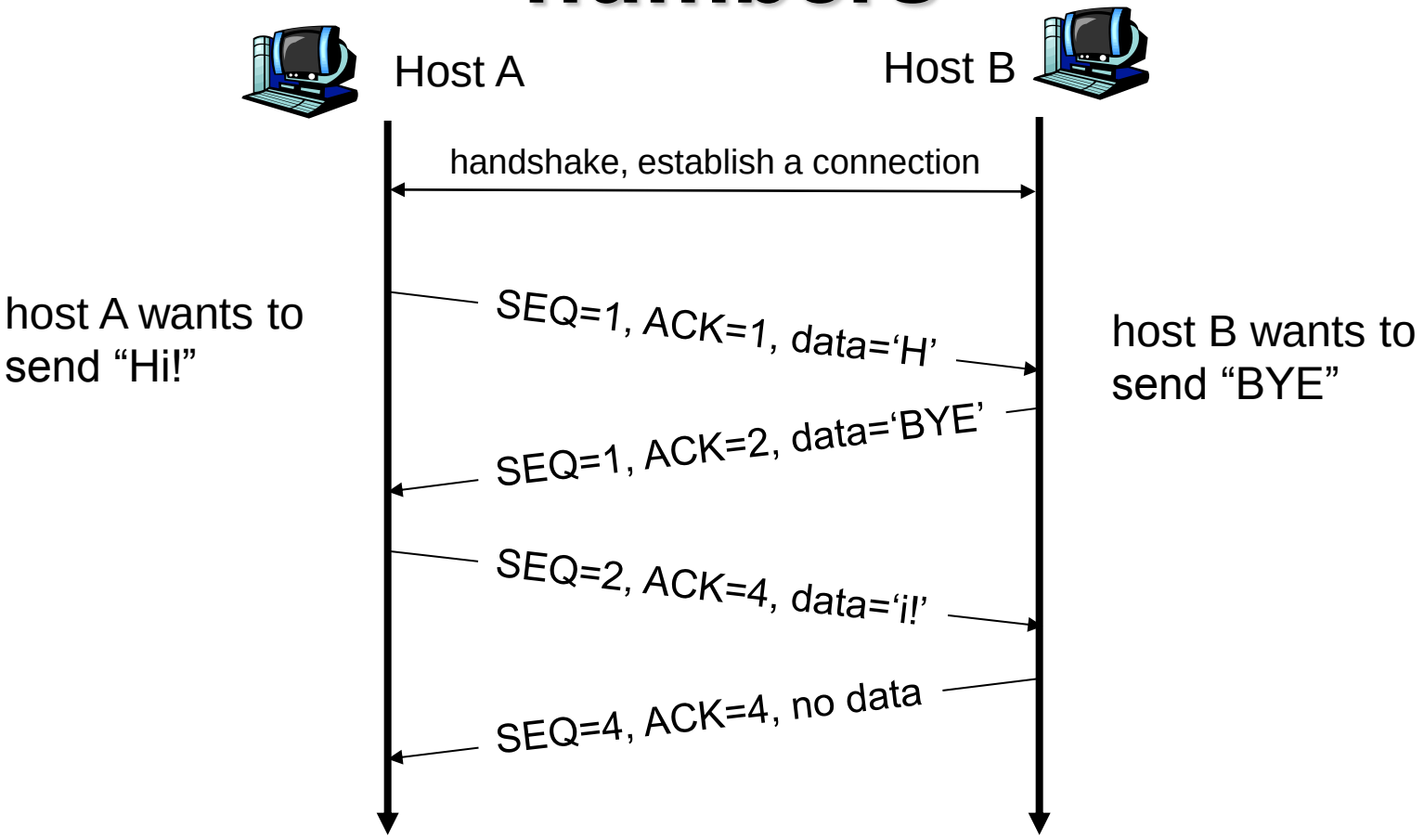
TCP Exploits

- Sometimes you may just want to kill (deny) a session.
- **TCP Exploits** use the TCP characteristics to launch attacks
  - Also use IP address spoofing to map packets to a TCP connection

TCP Basics

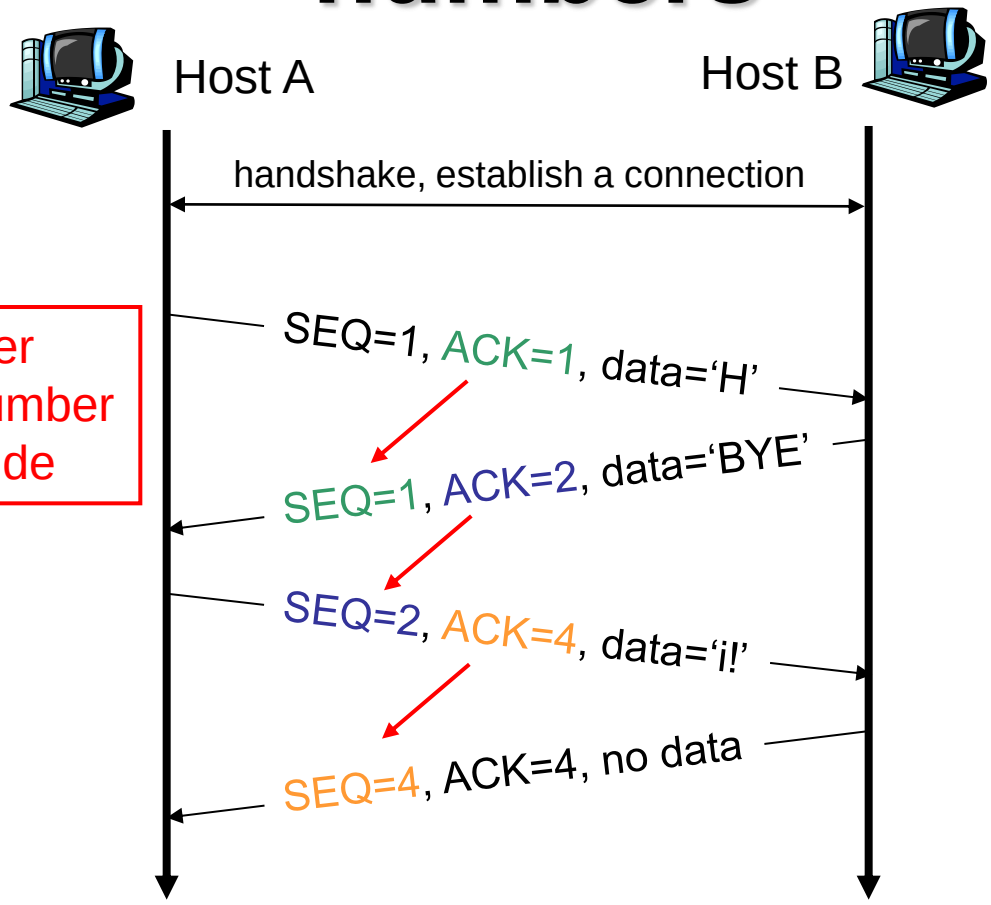
- TCP connections are identified by 4-tuples: (srcIP, srcPort, dstIP, dstPort)
- TCP views data as unstructured, but ordered, stream of bytes
- Sequence number for a segment:
  - byte-stream number of the first byte in the segment
- ACK number
  - sequence number of next byte expected from other side

TCP sequence numbers / ACK numbers



Example

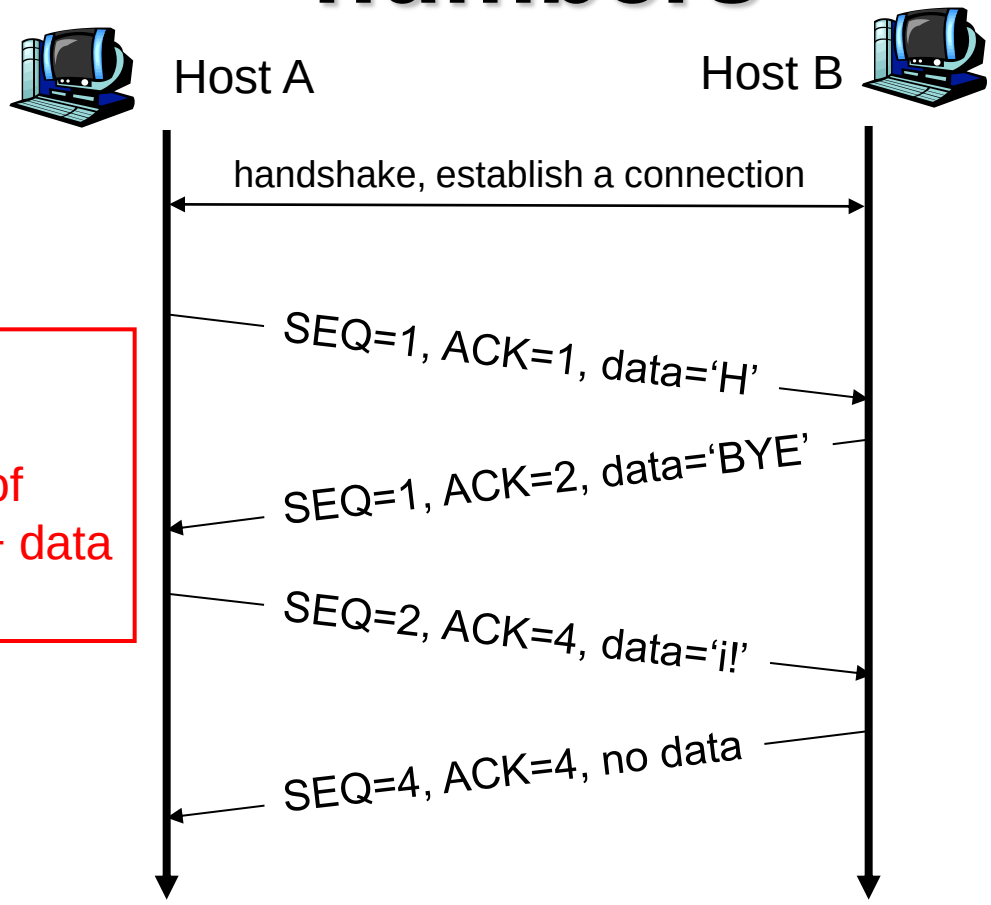
TCP sequence numbers / ACK numbers



Rule #1: seq number must follow ACK number reported by other side

Example

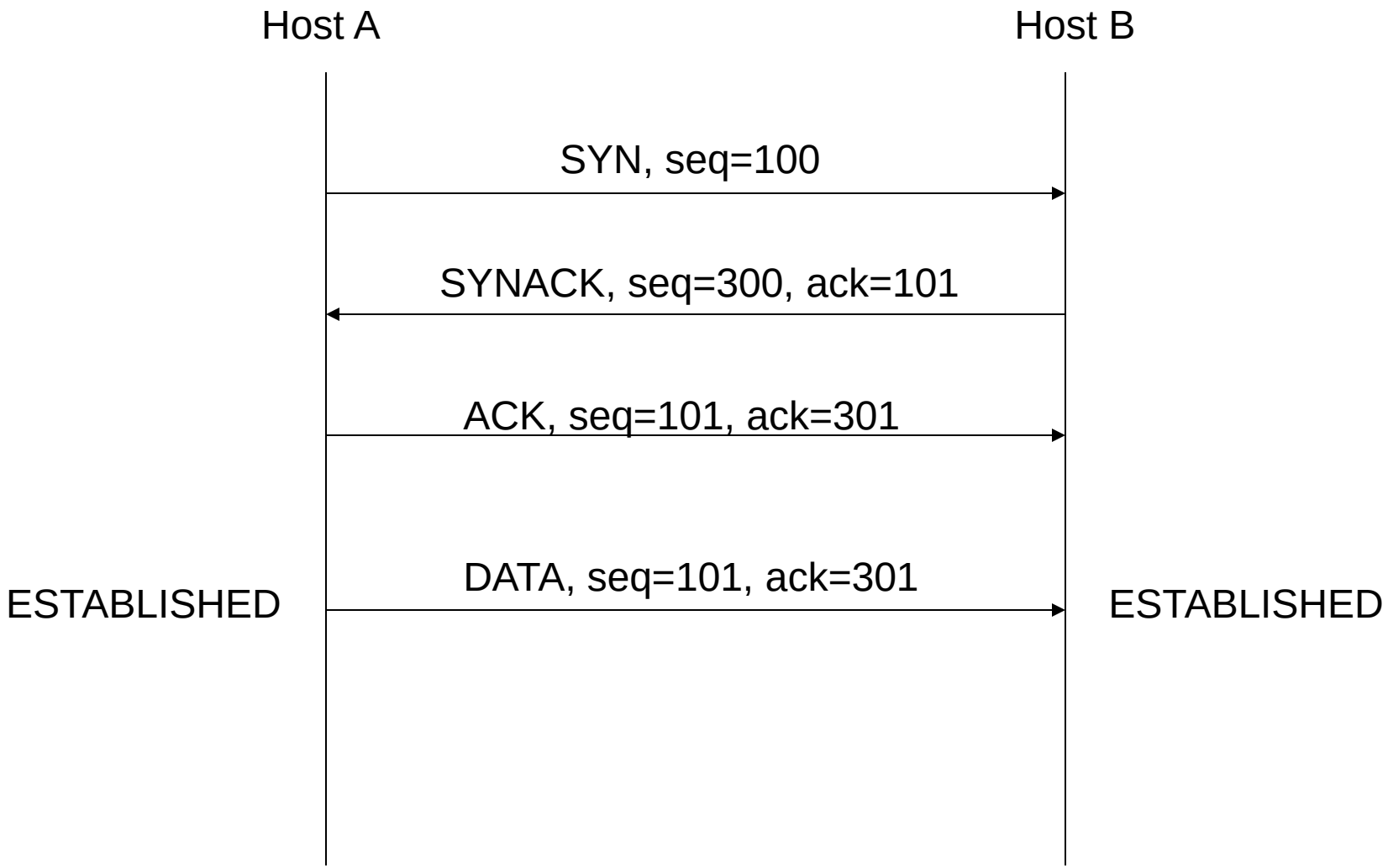
TCP sequence numbers / ACK numbers



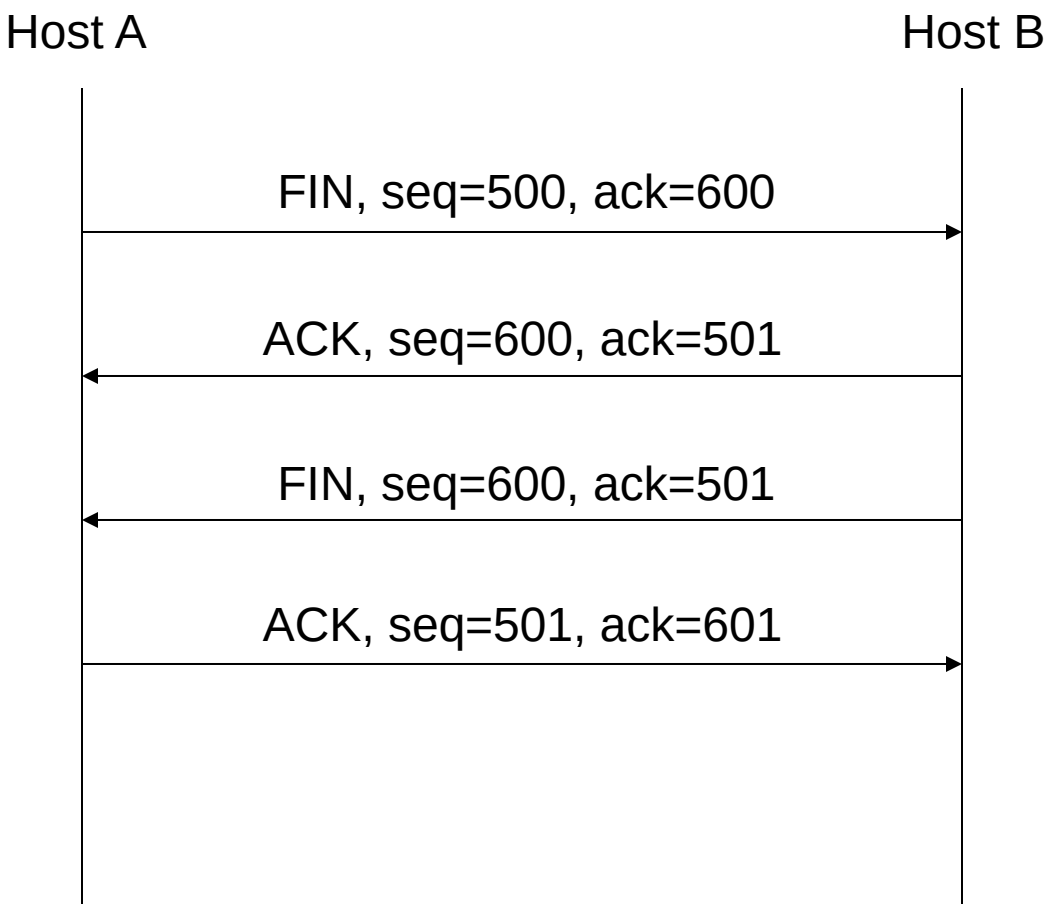
Rule #2:
ACK number = sequence number of received segment + data length in bytes

Example

TCP 3-Way Handshake



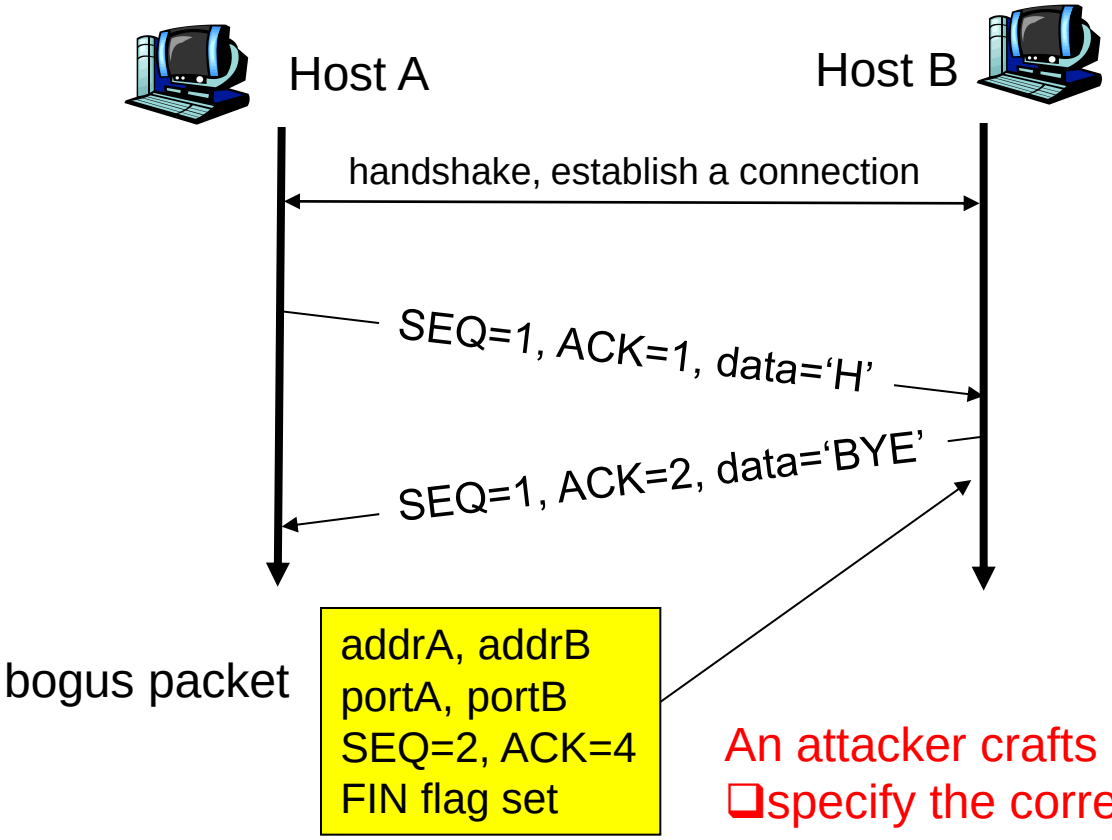
TCP Connection Closing



TCP Exploits

- TCP packets with the correct 4-tuple and sequence number will be accepted
- TCP exploits: craft a FIN (or RST) packet with the correct 4-tuple and sequence number to kill a connection

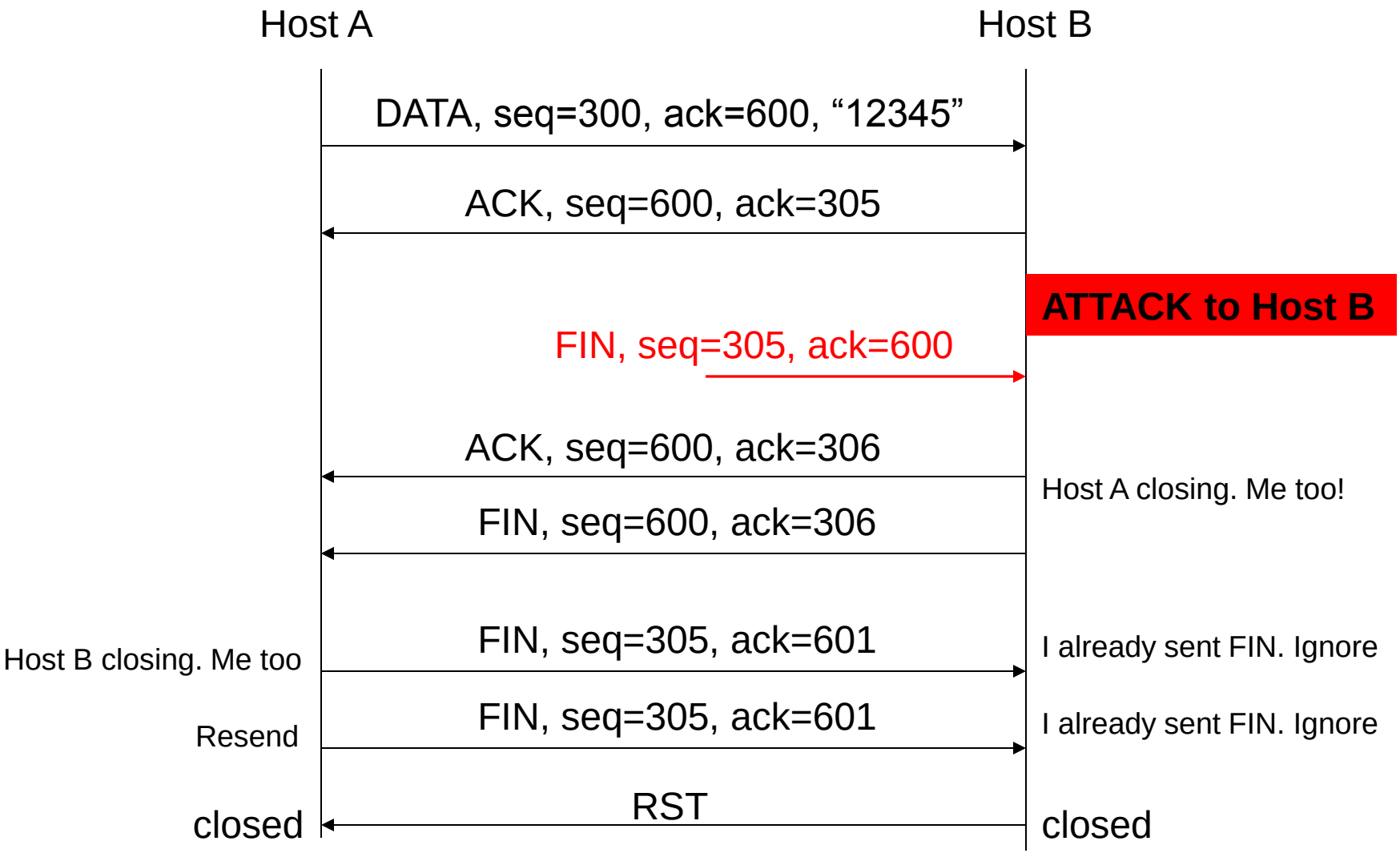
TCP Exploits: Main Idea



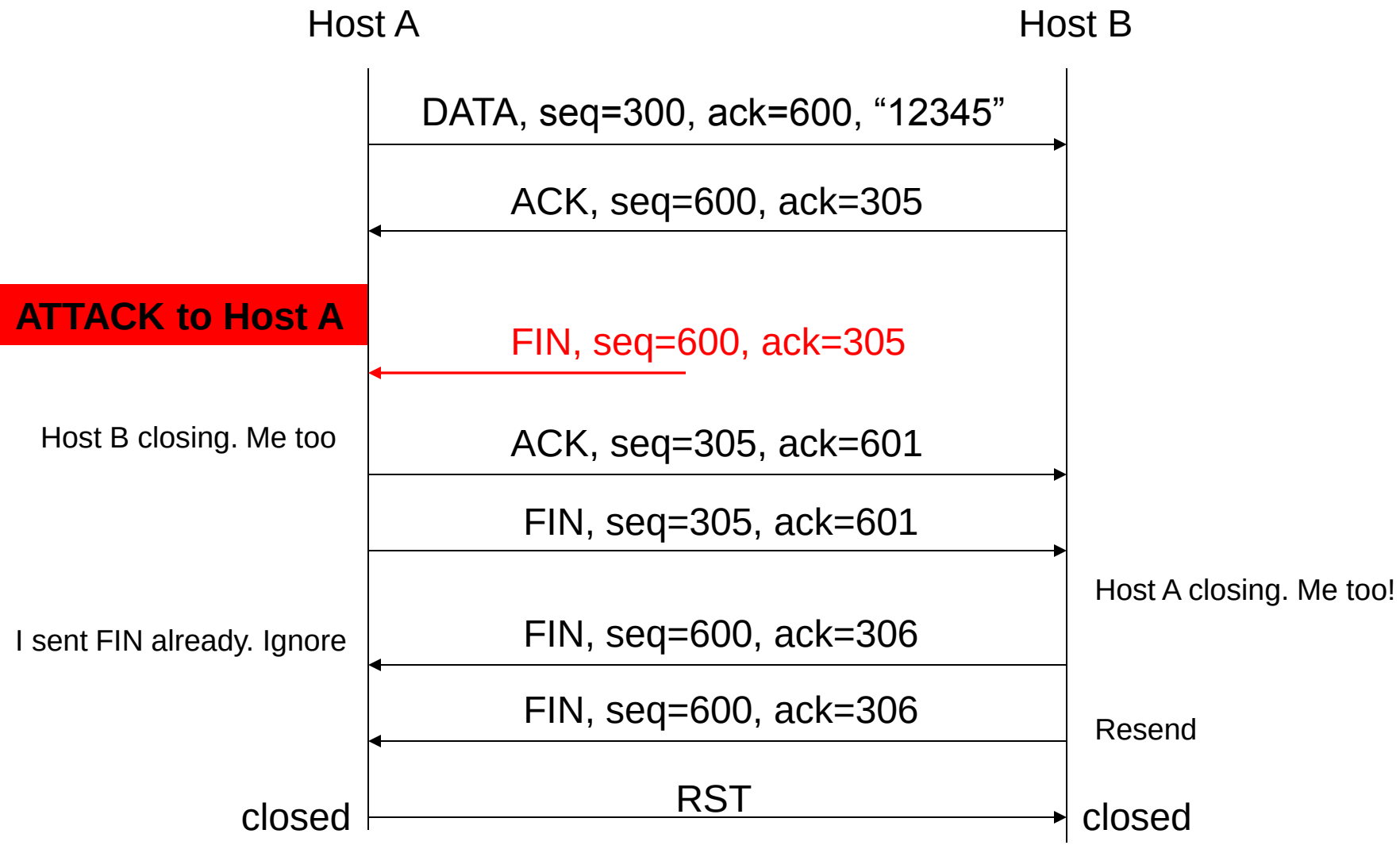
An attacker crafts a bogus packet:

- specify the correct 4-tuple so that host B maps the TCP connection
- specify the right seq/ack numbers so that the TCP connection responds
- specify the FIN flag to kill the connection

TCP Exploits – Hack Host B



TCP Exploits – Hack Host A



Comments

- Attackers may send the bogus packet later than the real sender
- May require some retries
- Suggestions:
  - Fire multiple packets with increasing sequence numbers
  - Carry 2 attacks in both ways simultaneously

Defense Against TCP Exploits

- TCP exploits work in the transport layer. Attacks work even you encrypt traffic in application layer (e.g., HTTPS, SSH)
- How to defend?
  - Use VPN (virtual private network): network-layer protection using IPSec

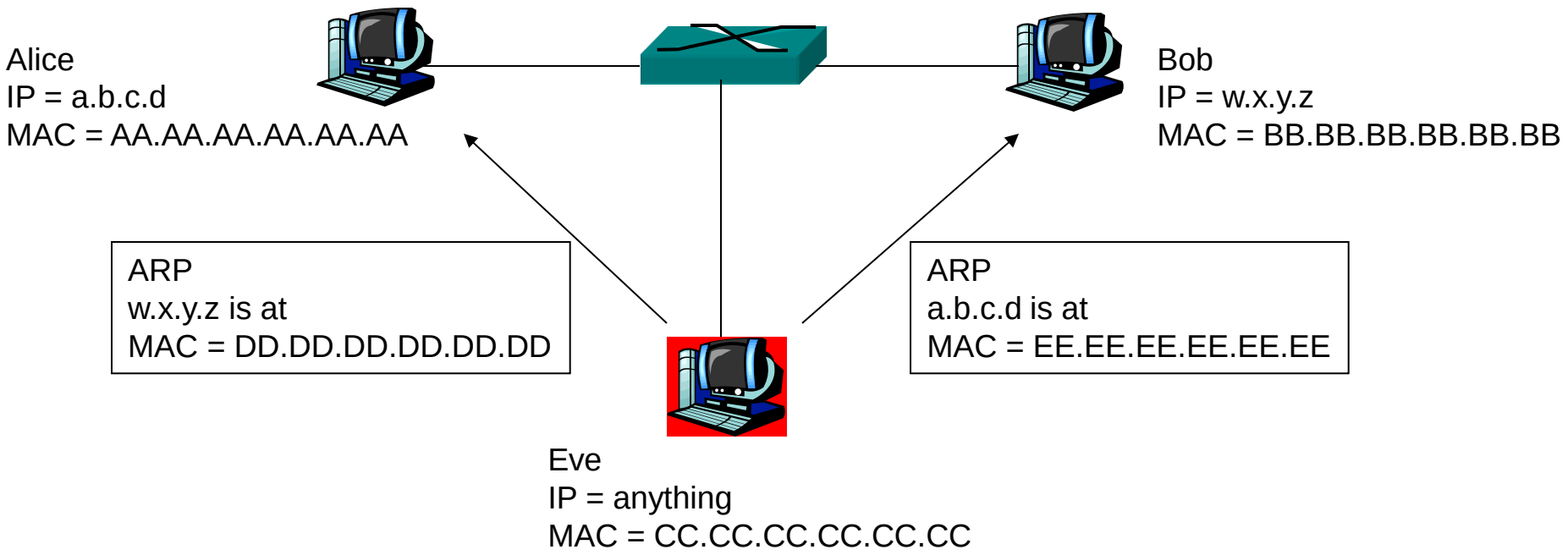
Roadmap

- Passive Sniffing
- Active Sniffing with ARP Spoofing
- TCP Exploits
- Session Hijacking
- Netcat

Session Hijacking

- Goal: steals an active session
- Tools: Hunt (or Ettercap)
- How it works:
  - Alice has an existing session with Bob
  - Eve sniffs the traffic between Alice and Bob (e.g., through ARP spoofing)
  - Eve injects traffic
  - Alice/Bob think Eve's traffic from Bob/Alice

Session Hijacking Scenario



- Suppose Eve sits in the same LAN as Alice and Bob
- Eve send spoofed ARP replies to Alice and Bob
- Alice and Bob's traffic is sent to Eve, who can now relay traffic.

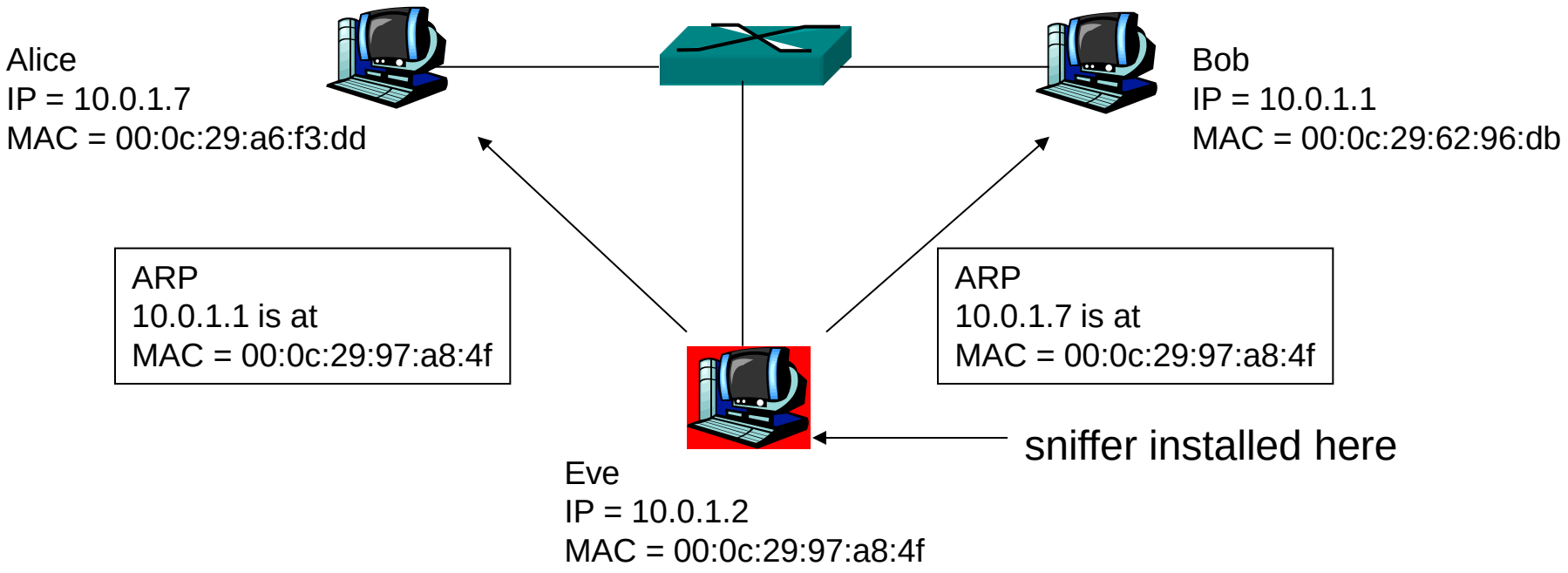
Session Hijacking Scenario

- Wait… doesn't the MAC address in the ARP replies need to be Eve's?
- You may use fake MAC addresses. Some switches will learn which port the fake MAC address comes from. When the switch receives frames destined for the fake MAC addresses, it will forward the frames to the port.

How to Use Hunt

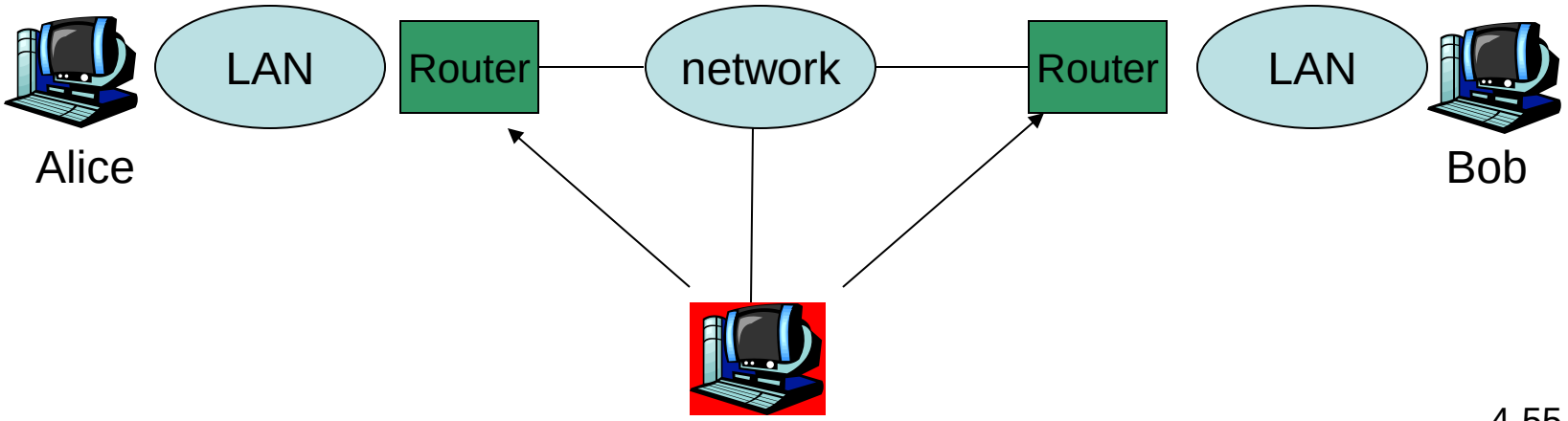
- Take-home exercises:
  - Learn how to launch the attack by yourself using hunt, and explore other features.
- Source:
  - http://www.packetstormsecurity.org/sniffers/hunt/hunt-1.5.tgz
- See hunt.pcap
  - Use hunt to launch ARP spoofing
  - Use hunt to hijack a telnet session

In hunt.pcap


Alice
IP = 10.0.1.7
MAC = 00:0c:29:a6:f3:dd
Bob
IP = 10.0.1.1
MAC = 00:0c:29:62:96:db
ARP
10.0.1.1 is at
MAC = 00:0c:29:97:a8:4f
ARP
10.0.1.7 is at
MAC = 00:0c:29:97:a8:4f
sniffer installed here
Eve
IP = 10.0.1.2
MAC = 00:0c:29:97:a8:4f

Session Hijacking – not in the same LAN

- Session hijacking works even if Alice, Bob, and Eve are on different LANs
- Eve can send gratuitous ARP messages to the routers that have Alice and Bob's traffic.



Session Hijacking Defenses

- Use techniques against sniffing and spoofing
  - Even if an attacker hijacks your session, no information is leaked
- Pay attention to certificate warnings
  - Make sure the origin real, rather than monkey in the middle
- Can we use some intrusion detection techniques, e.g., identify hosts that send too many gratuitous ARPs?

Roadmap

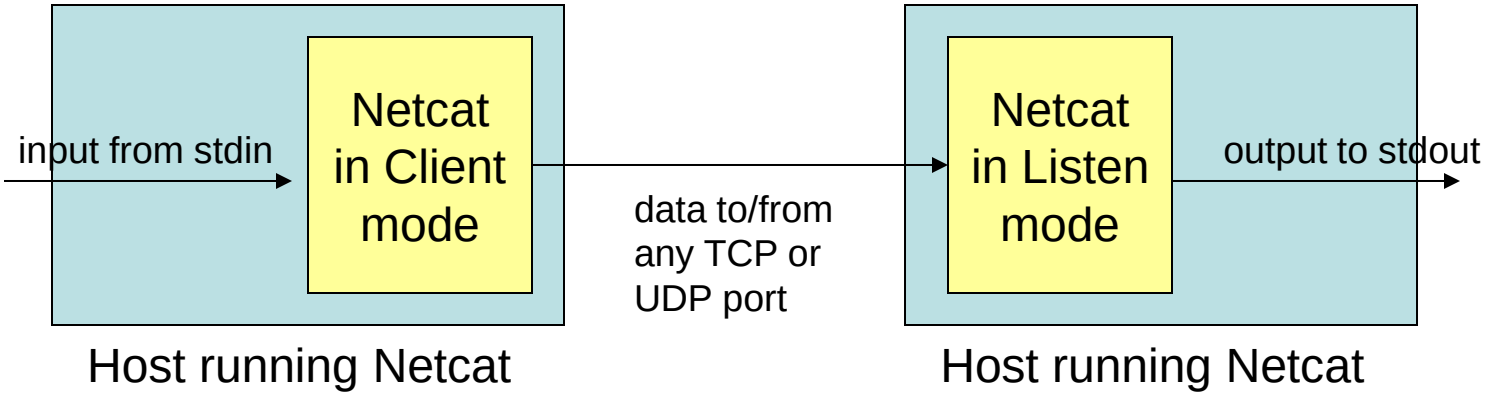
- Passive Sniffing
- Active Sniffing with ARP Spoofing
- TCP Exploits
- Session Hijacking
- Netcat

Netcat

- **Netcat** is a general-purpose network tool for interacting with systems across a network
  - used by attackers and system administrators
  - the Swiss Army knife of network tools
- Use netcat to create a backdoor in a compromised machine.
- Source: http://netcat.sourceforge.net/

Netcat

- Netcat is a network version of “cat” command
- Send/receive data to/from any TCP/UDP port
- Two modes: client mode and listen mode



BONUS Slide

Netcat for File Transfer

Destination machine receiving file

```
$ nc -l -p 1234 > [file]
```

listen mode

on local port 1234

dump data to a file

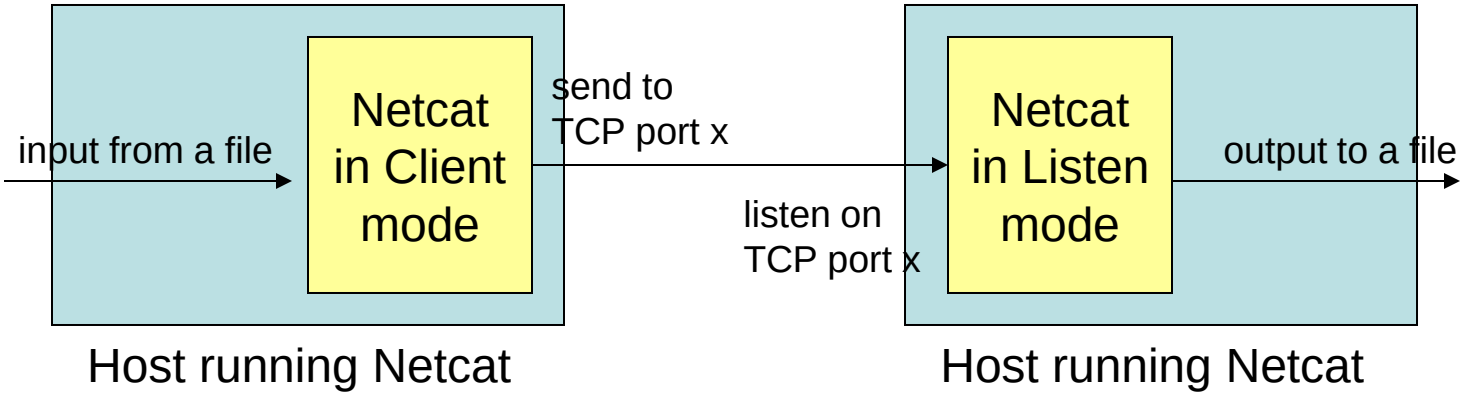
Source machine sending file

```
$ nc [remote machine] 1234 < [file]
```

client mode (default)

on remote port 1234

input file



- Pushing a file across the network

BONUS Slide

Netcat for File Transfer

Source, offering file

```
$ nc -l -p 1234 < [file]
```

listen mode

on local port 1234

file to send

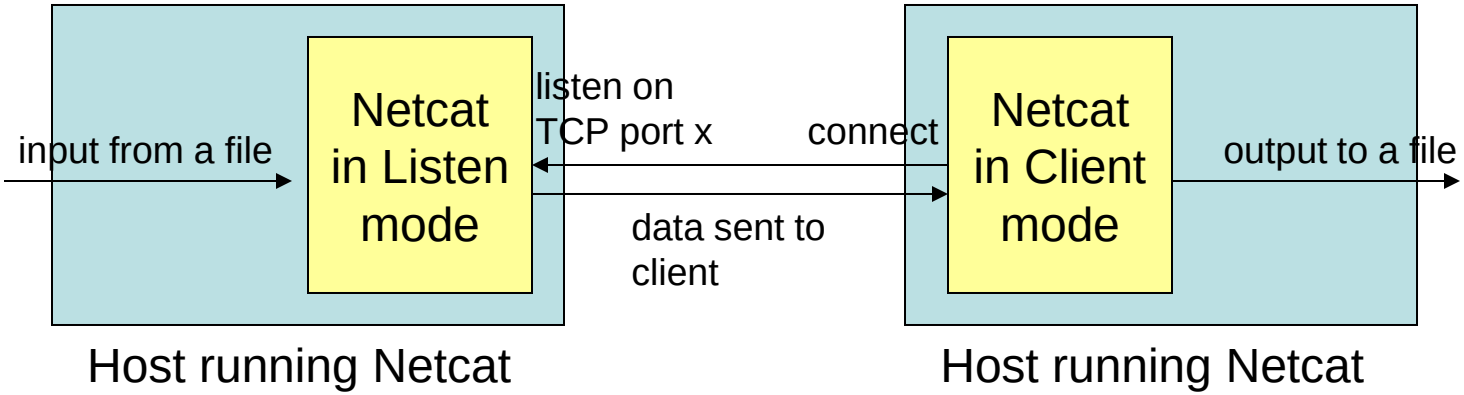
Destination, pulling file

```
$ nc [remote machine] 1234 > [file]
```

client mode (default)

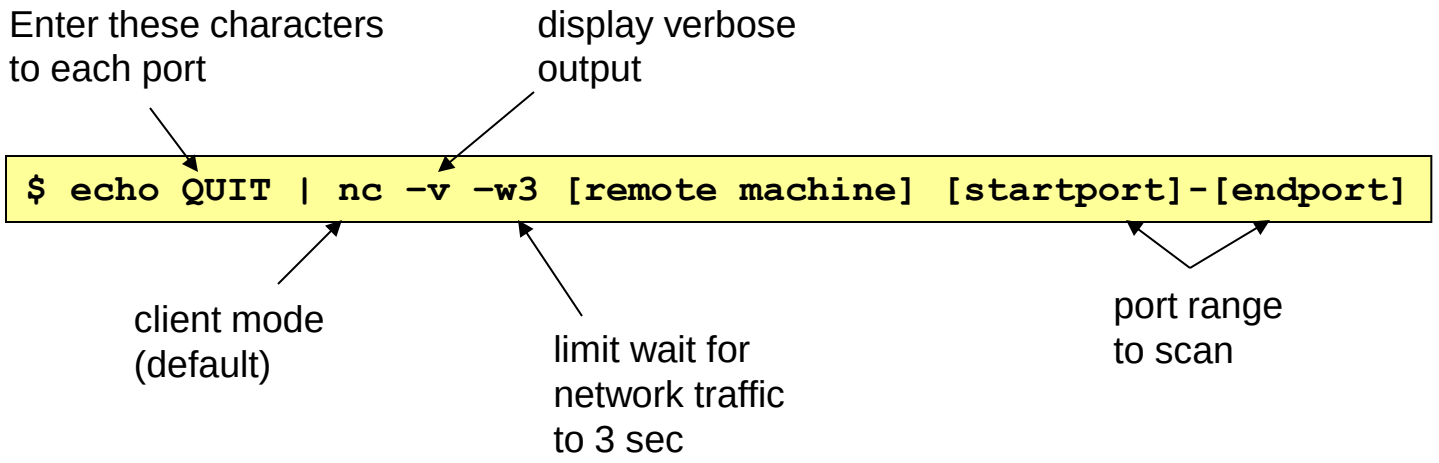
on remote port 1234

dump to file



- Pulling a file across the network

Netcat for Port Scanning

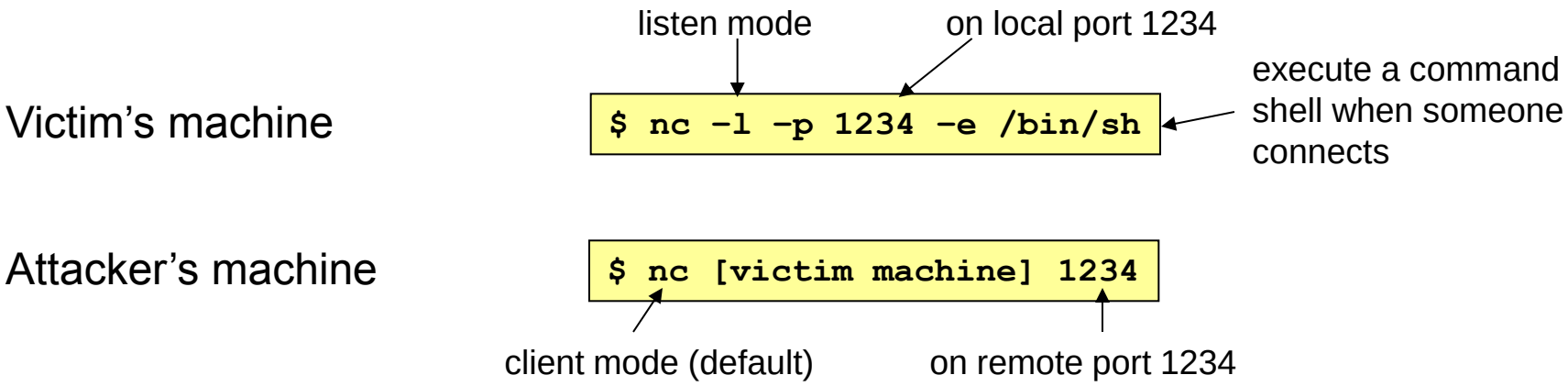


- Netcat can connect to many ports
- Nmap offers more features in port scanning (http://nmap.org)

BONUS Slide

Create a Passive Backdoor Command Shell

- Suppose Trudy gains access to a machine
  - How? e.g., by tricking victims to run Trojan horse…



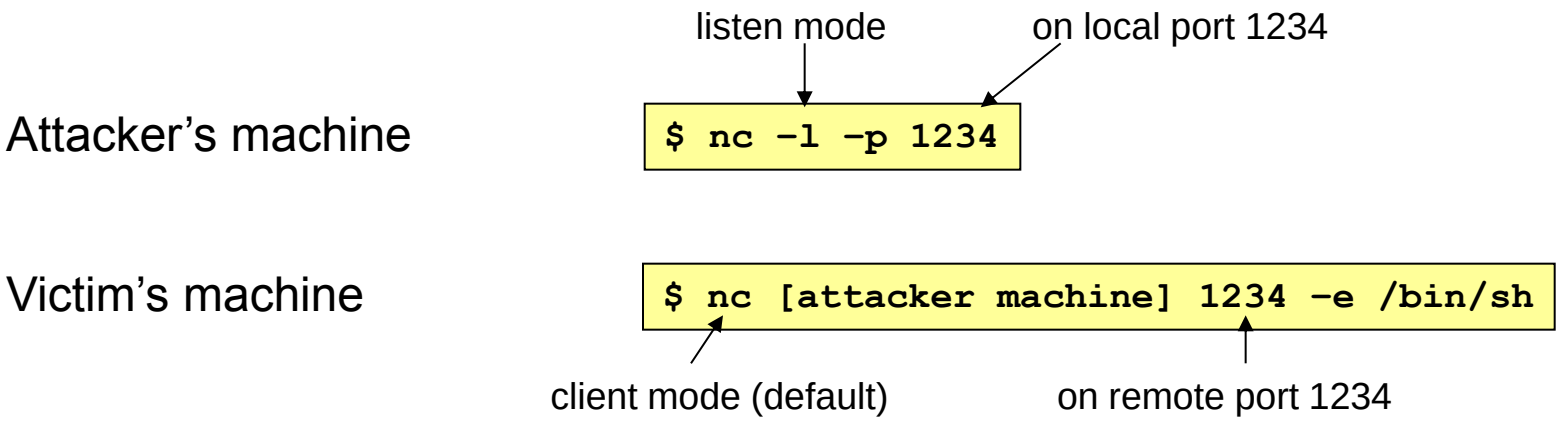
BONUS Slide

Actively Push a Backdoor Command Shell

- Passive backdoors may not work if a firewall blocks inbound connections on local ports
- To get around this, Trudy creates a listen port on her own machine, and gets the victim to push its own command shell to Trudy's machine
  - called reverse shell or shell shoveling

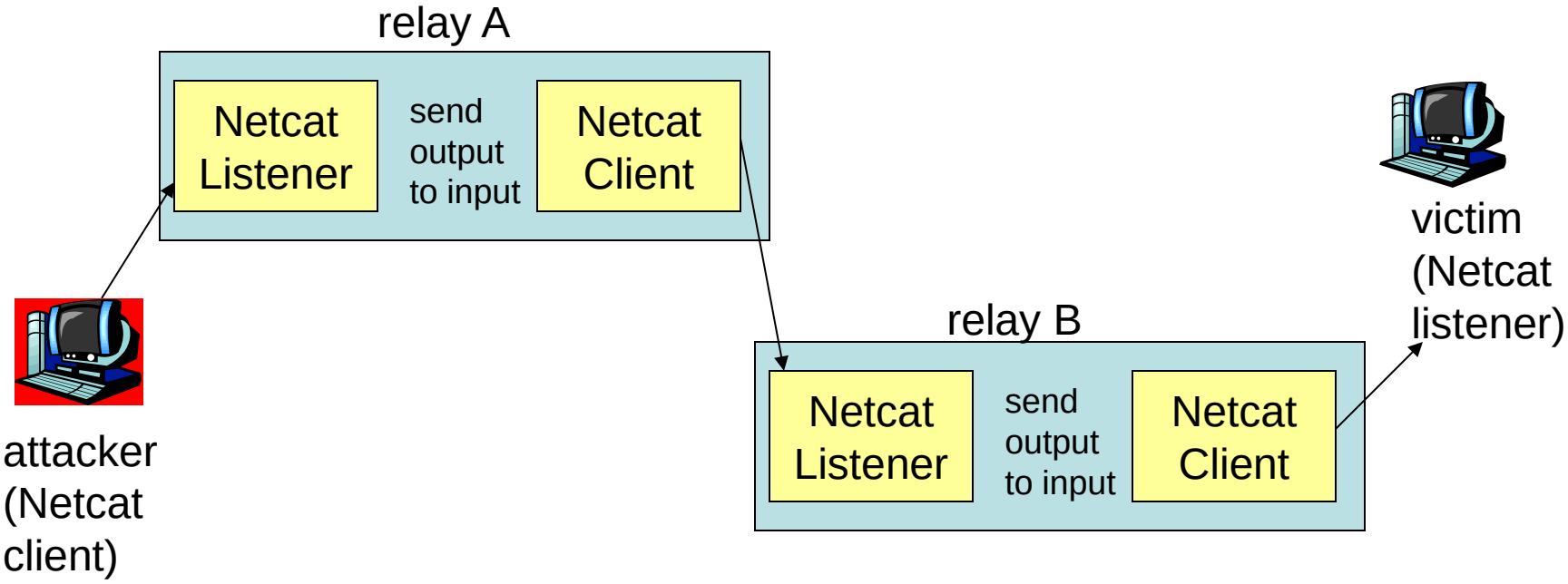
BONUS Slide

Actively Push a Backdoor Command Shell



- This attack works if firewall allows outbound connections.

Relaying Traffic with Netcat



- Set up an attack channel over multiple relays. Make traceback difficult.

BONUS Slide

Netcat Defenses

- Prevent Netcat file transfers
  - by configuring firewalls to limit incoming/outgoing traffic
- Securing against port scanning
  - Configure the system with minimum number of open ports
- Blocking arbitrary connections
- Apply patches to avoid unexpected executions of netcat
- Can we use some intrusion detection techniques to identify unexpected traffic?

References

- We discuss traditional, but practical network attacks that exploit the fundamental limitations of network protocols to gain malicious access
- References:
  - Skoudis and Liston, “Counter Hack Reloaded”, Chapter 8.