Enrico RUSSO
Università di Genova

Network Security I

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

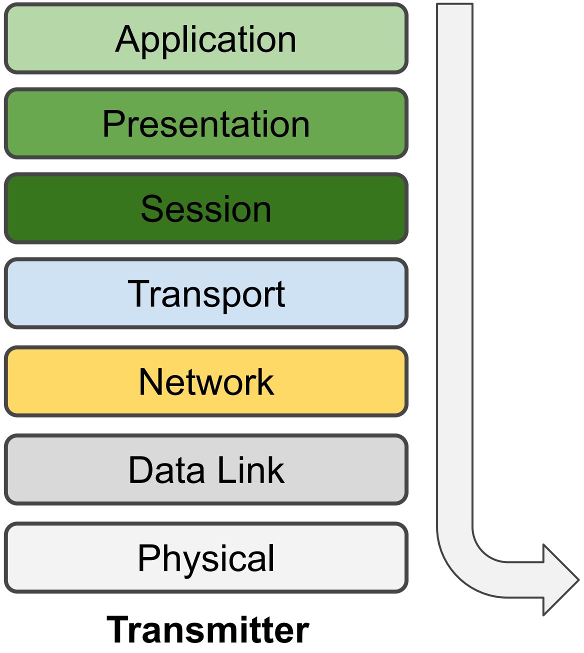
ISO/OSI and TCP/IP

- ISO/OSI and TCP/IP represent the reference models for communication between different computers in the network. They both use a **layered** model.
  - Separate networking functions into logical smaller pieces: network problems can more easily be solved through a **divide-and-conquer** methodology.
  - Provide **modularity** and **clear interfaces**: they allows the standardization of interactions among devices.
  - Allow **extensibility**: new network functions are generally easier to add to a layered architecture.
- ISO/OSI model evolved as a **theoretical** model.
- TCP/IP as a **practical** model, founded on widely used implementation of network functions.

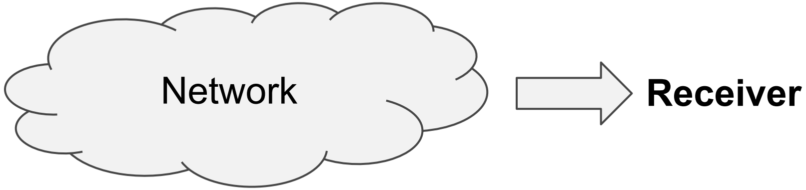
OSI Layers

Application	It provides the services to the user
Presentation	It is responsible for the formatting of information (e.g., compression and encryption)
Session	It is responsible for establishing, managing, and terminating sessions
Transport	It provides message delivery from process to process
Network	It is responsible for moving the packets form source to destination
Data Link	It combines bits into a structure of data and provides their error-free transfer
Physical	It provides a physical medium through which bits are transmitted

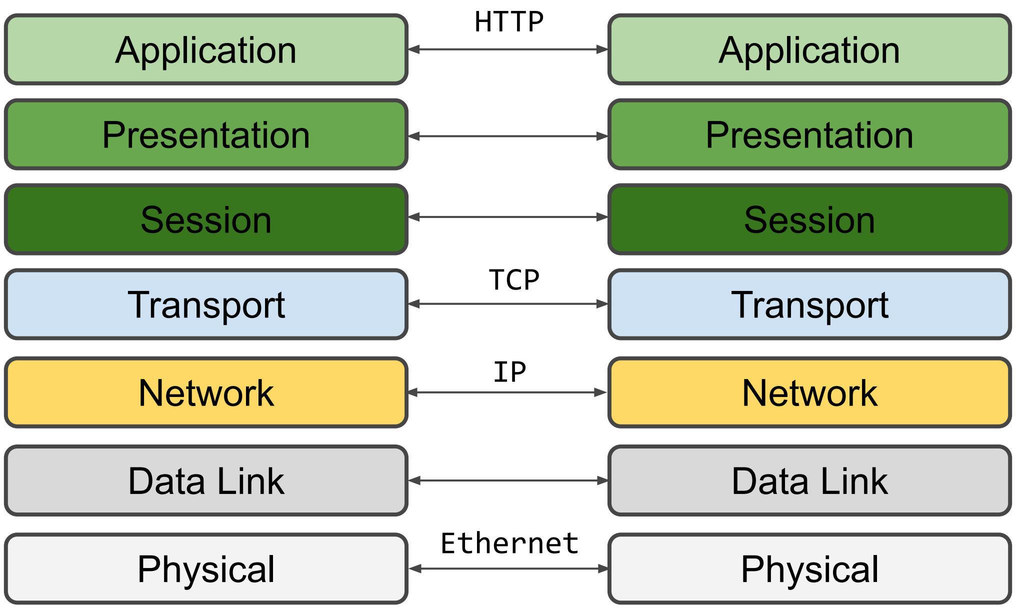
OSI Layers: data transfer



- The initial data transfer begins at the application layer of the transmitter
- Each layer can communicate just with the layers directly above and below it
- The communication going from top to bottom on the transmitter device and then from bottom to top when it reaches the receiver

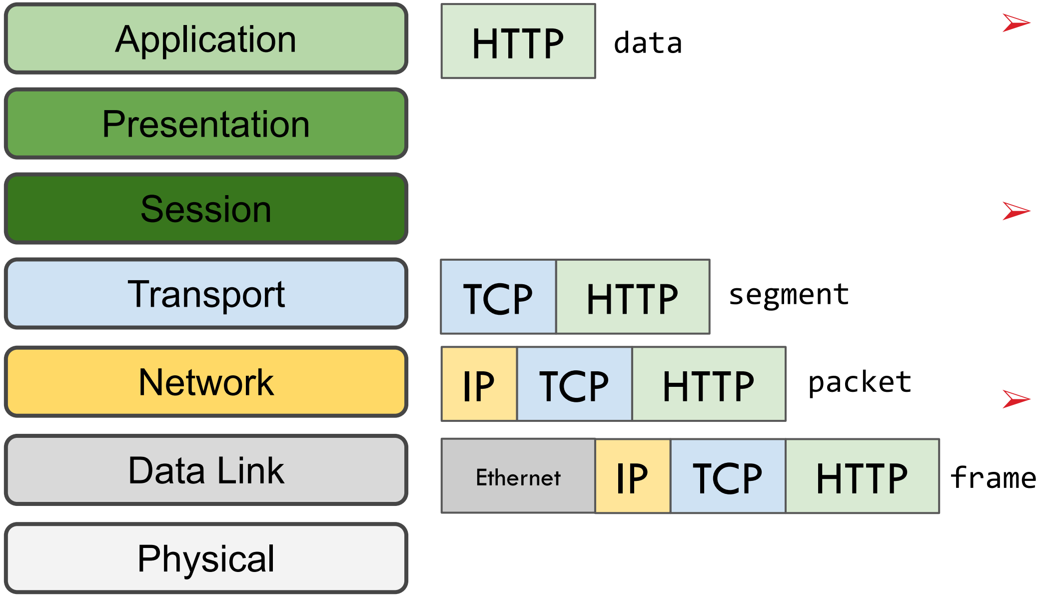


OSI Layers: protocols



- The model itself does not provide specific methods of communication
- Actual communication is defined by various *protocols*
- A protocol is a **standard procedure and format** that two data communication devices must understand, accept and use to be able to talk to each other

OSI Layers: Protocols Data Unit (PDU)



- The protocols at different layers exchange data with the aid of *data encapsulation*
- Each layer is responsible for adding a header or a footer to the data being transferred
- The encapsulation process creates a *Protocol Data Unit* (PDU), which includes the data being sent and all header or footer information added to it

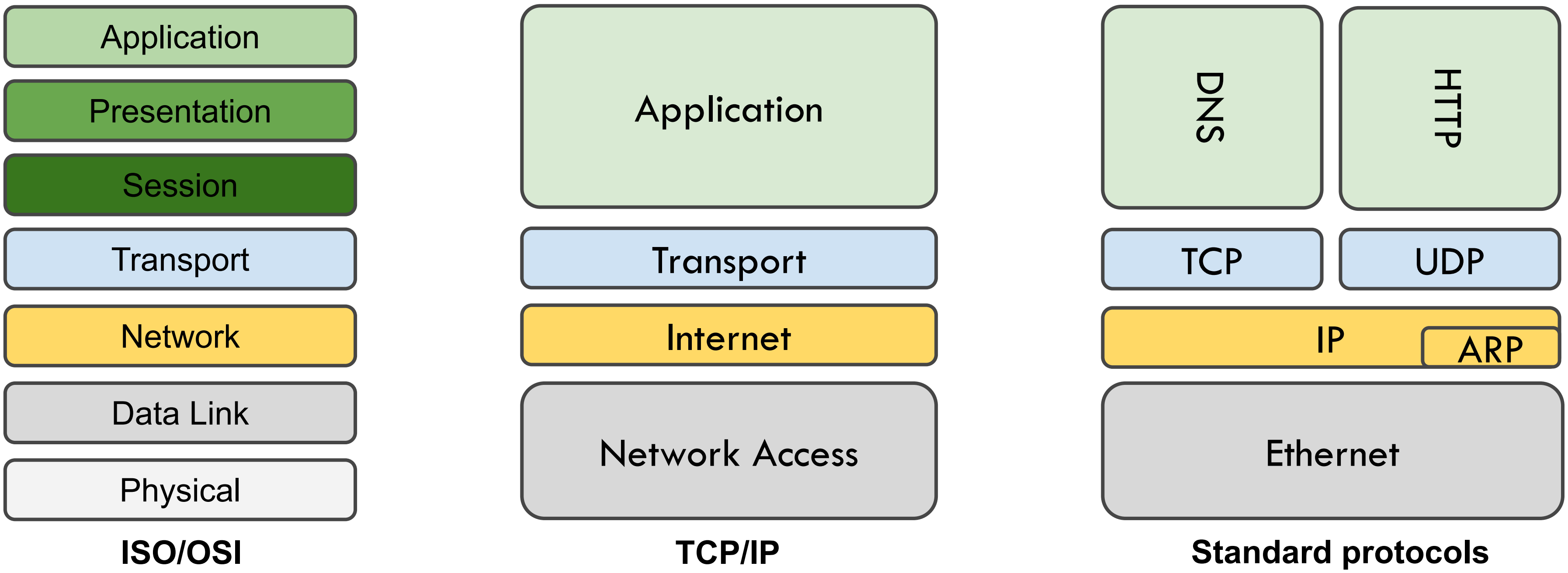
 Rel. 14.03.2021

TCP/IP

TCP/IP provides an alternative model used for the description of all network communications.

- is a four-layer model
- is based on standard protocols that the Internet has developed, and the name refers to the two widely used ones:
  - **Transmission Control Protocol** (TCP) which also implements the Transport layer of ISO/OSI model
  - **Internet Protocol** (IP) which also implements the Network layer of ISO/OSI model

TCP/IP model



Application
Presentation
Session
Transport
Network
Data Link
Physical
ISO/OSI
Application
Transport
Internet
Network Access
TCP/IP
DNS
HTTP
TCP
UDP
IP
ARP
Ethernet
Standard protocols

The client-server model

- TCP/IP relies on the **client-server** model for enabling the process communication between network nodes.
  - It is a relationship in which one program (*client*) requests a service or resource from another program (*server*).
  - The client needs to know of the existence of and the address of the server.
  - The server does not need to know the address of (or even the existence of) the client prior to the connection being established.

Ethernet

Ethernet is a broadly deployed layer 2 protocol.

- Encapsulate data and transmit them in the form of *frames*
- Frames leverage the Media Access Control (MAC) addresses
  - Every Ethernet device (e.g., a server, a switch, or a router) has a unique MAC address on its local network
  - A *Frame* includes the MAC address of the destination interface on the target system as well the MAC address of the source interface on the sending system

*https://www.wireshark.org/tools/oui-lookup.html

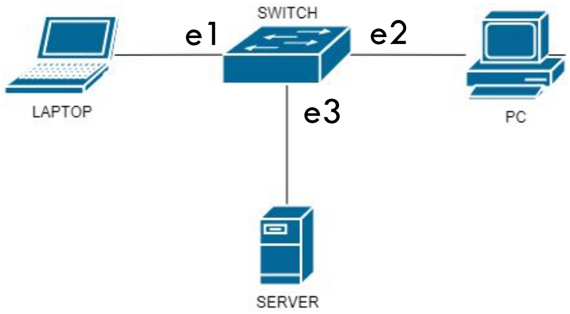
Bridges and Switches

Devices providing interconnectivity at Layer 2 are called *(Transparent) Bridges or Switches.*

- They analyze all frames received, find the destination MAC address, and forward them to the appropriate port.
- To determine where to forward the traffic, they use a special table (MAC address table).

A basic switched network

- A switch device provides connection to a number of common devices.
- Let's assume that all the devices be powered on but have not sent any traffic.
- In this case, the MAC address table of the switch would be empty.

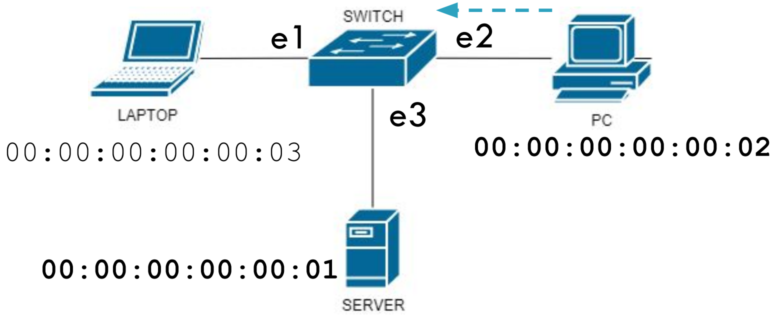


MAC address table (switch)

MAC address	Port

A basic switched network

- PC wants to send traffic to SERVER that has MAC address `00:00:00:00:00:01`
  - Creates a frame containing `00:00:00:00:00:02` as the source address and `00:00:00:00:00:01` as the destination address.
  - Sends it off toward the switch.

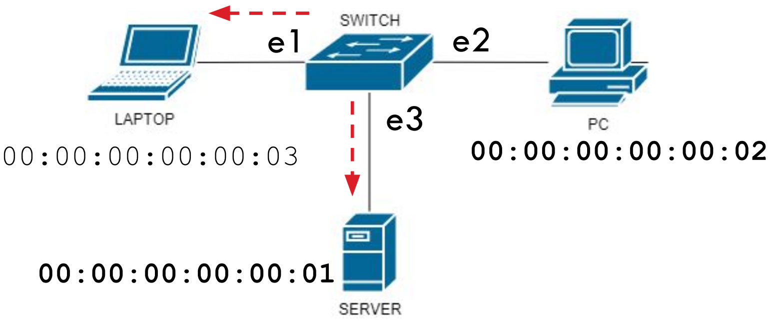


MAC address table (switch)

MAC address	Port

A basic switched network

- The switch receives the traffic
  - Creates a new entry in its MAC address table for PC MAC address (PC → e2)
  - Performs a lookup on its MAC address table to determine whether it knows which port to send the traffic to
  - Since no matching entries exist in the switch's tables, it would **flood** the frame out all of its interfaces except the receiving port (**broadcast**).

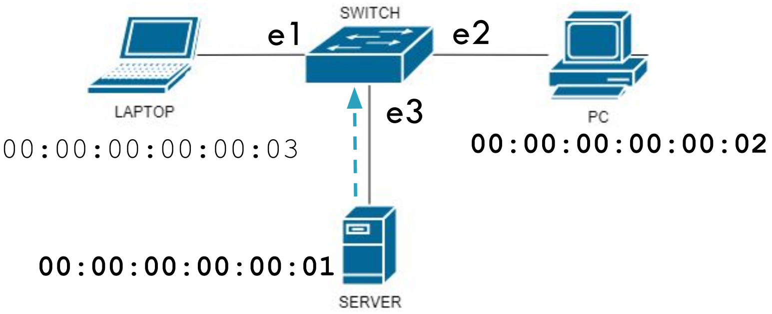


MAC address table (switch)

MAC address	Port
00:00:00:00:00:02	e2

A basic switched network

- The broadcast forwards the frame also to the target server.
- (Assuming that the server wants to respond to PC) It sends a new frame back toward the switch containing `00:00:00:00:00:01` as the source address and `00:00:00:00:00:02` as the destination address.
- The switch would receive the frame and create a new entry in its MAC address table for the Server MAC address (Server → e3).

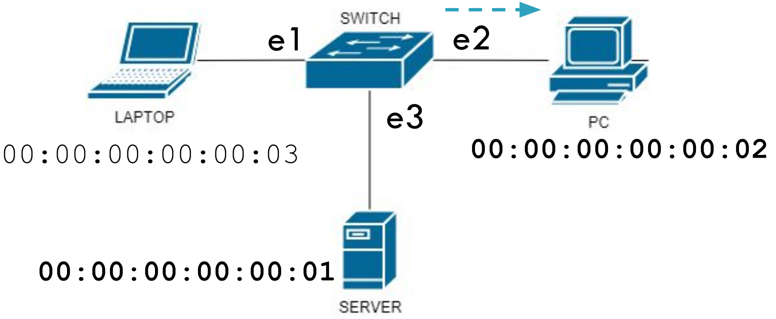


MAC address table (switch)

MAC address	Port
00:00:00:00:00:02	e2
00:00:00:00:00:01	e3

A basic switched network

- Switch performs a lookup of its MAC address table to determine whether it knows which port to send the server frame to.
- In this case, it does, so it sends the return traffic out only its e2 port (PC), without flooding.



MAC address table (switch)

MAC address	Port
00:00:00:00:00:02	e2
00:00:00:00:00:01	e3

Internet Protocol (IP)

The most significant protocol at layer 3 is the *Internet Protocol* or IP

- The standard for routing packets across interconnected networks (hence, the name internet)
- Encapsulate data and pass that data in the form of *packets*

IP addressing

- An Internet Protocol address is also known as an **IP address**.
- A numerical label which assigned to each device connected to a computer network that uses the IP for communication.
- Two versions: IPv4 and IPv6
  - IPv6 is the new version that is being deployed to fulfill the need for more Internet addresses.
  - In this module, we focus on IPv4 (currently the most widely used).

IP addressing

- IPv4 address
  - 32 bits
  - Grouped 8 bits at a time (octet)
  - Each of the four octets is separated by a dot and represented in decimal format (dotted decimal notation)

11000000 10101000 01100100 11001000

192 . 168 . 100 . 200

IP addressing - Home addressing

192.168.100.200
(host address)

192.168.100
(network address)

2^{32}

IPv4
available
addresses

35
(house number)

via Dodecaneso
(street name)

IP address and Netmask

- An IP address has two components: a *network* component (street name), and a *host* component (house number)
- The purpose of the *netmask* is to split the IP address into the two components
- When you combine, using a logical AND, the IP address and the netmask you reveal the network component

	11000000	10101000	01100100	11001000	address
	192 .	**168** .	**100** .	**200**	
	11111111	11111111	111111111	00000000	netmask (/**24**)
	255 .	**255** .	**255** .	**0**	
network	11000000	10101000	01100100	00000000	host
	192 .	**168** .	**100**		

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

Reserved IP addresses

- In every network, two addresses are used for special purposes. <u>These addresses are not available for nodes</u>
- **Network address**: is the first address in the network (all the host bits are 0) and it is used for identifying the network
- **Broadcast address**: is the last address in the network (all the host bits are 1). An IP packet having the broadcast address as the destination address is sent to all nodes of the IP network

11000000 10101000 01100100 11001000 address

192 . 168 . 100 . 200

11111111 11111111 111111111 00000000 netmask (**/24**)

255 . 255 . 255 . 0

11000000 10101000 01100100 **00000000** network address

192 . 168 . 100 . 0

11000000 10101000 01100100 **11111111** broadcast addr.

192 . 168 . 100 . 255

Default Netmasks

- Default netmasks have all ones (255) or all zeroes (0) in an octet

Address Class	Total # Of Bits For Network ID / Host ID	Default Subnet Mask			
Class A	8/24	255	0	0	0
Class B	16/16	255	255	0	0
Class C	24/8	255	255	255	0

Private IP addresses

Private IP addresses are **not routed on the Internet,** and traffic cannot be sent to them from the Internet

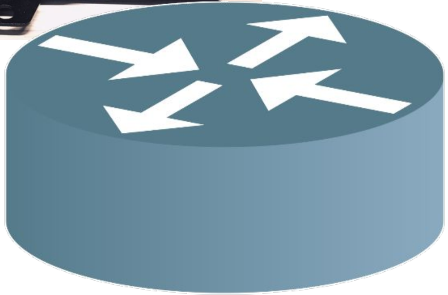
- They are supposed to work within the local network, only.
  - Range from 10.0.0.0 to 10.255.255.255 — a 10.0.0.0 network with a 255.0.0.0 or an /8 (8-bit) mask
  - Range from 172.16.0.0 to 172.31.255.255 — a 172.16.0.0 network with a 255.240.0.0 (or a 12-bit) mask
  - A 192.168.0.0 to 192.168.255.255 range, which is a 192.168.0.0 network masked by 255.255.0.0 or /16

IP Routing

- IP routing is the process of sending packets from a host on one network to another host on a different remote network
  - Nodes examine the destination IP address of a packet, determine the next-hop address, and forward the packet
  - Nodes use **routing tables** to determine a next hop address to which the packet should be forwarded

Router

- A router is the Layer 3 device that forwards data packets between computer networks.
- A router is connected to two or more data lines from different IP networks.

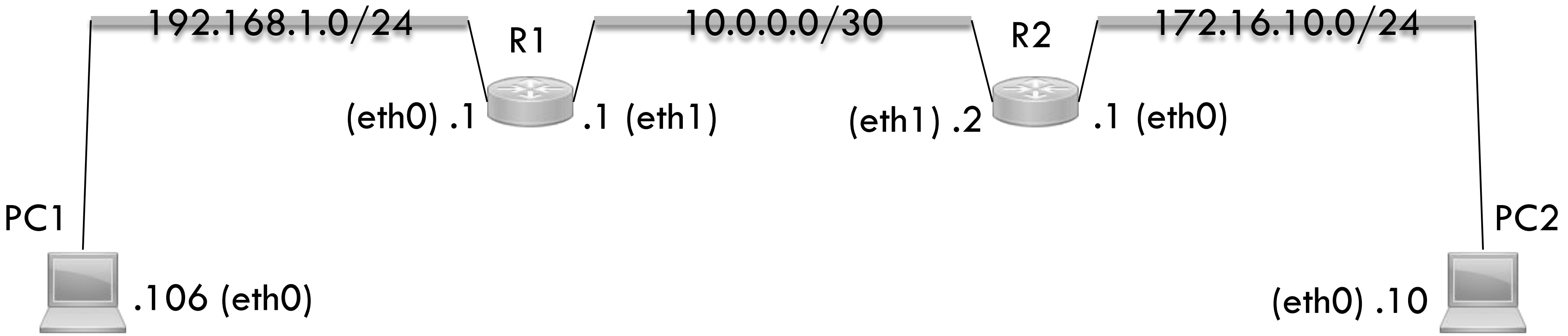


Internetworking: Routing Table

A routing table is used by nodes to determine the path to the destination network

- Each routing table consists of the following entries:
  - **Network destination and subnet mask** – specifies a range of IP addresses
  - **Remote router** – IP address of the router used to reach that network
  - **Outgoing interface** – outgoing interface the packet should go out to reach the destination network

Routing tables (example)



TCP vs UDP

- TCP and UDP are the most common Layer 4 protocols
  - TCP first creates a connection before any message is sent, whereas UDP does not
  - While both do error checking by checksums, UDP won't recover from one. TCP includes error recovery, thanks to acknowledgments
  - TCP rearranges data packets in the specific order while UDP protocol has no fixed order
  - Since UDP has no connection establishment, no connection state, and small packet header overhead is simpler and faster than TCP
  - UDP is commonly used for applications that are "lossy" (can handle some packet loss), such as streaming audio and video.

Layer 4 addressing: ports

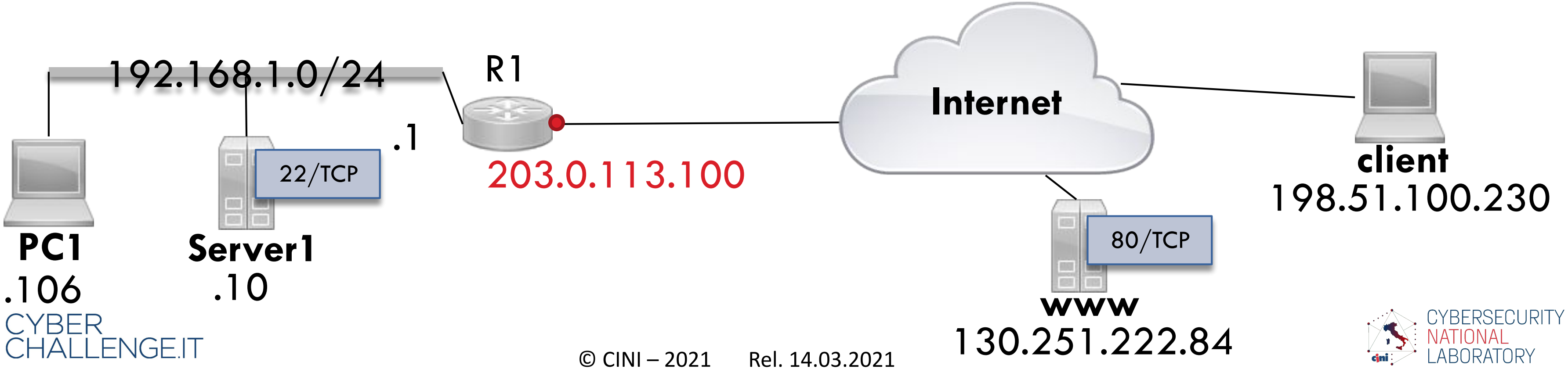
- Layer 4 is in charge of the **process-to-process** communication. Transmitter and receiver are identified using **ports**
  - 16-bit unsigned integer (0-65535, 0 reserved)
    - **Well-known ports** (0-1023): used by system processes that provide widely used types of network services (require superuser privileges)
    - **Registered ports** (1024-49151): assigned by a central authority (the Internet Assigned Numbers Authority, IANA) for specific services
    - **Ephemeral ports** (49152–65535): contain dynamic or private ports that cannot be registered by IANA

Layer 4 addressing: ports

- The use of well-known and registered ports allows the requesting process to easily locate the corresponding server application processes on other hosts
  - For example, a web browser knows that the web server process listens on port 80/TCP
- Despite these agreements, <u>any service can listen on any port</u>
  - For example, a web server process can listen on port 8080/TCP instead of the well-known one.

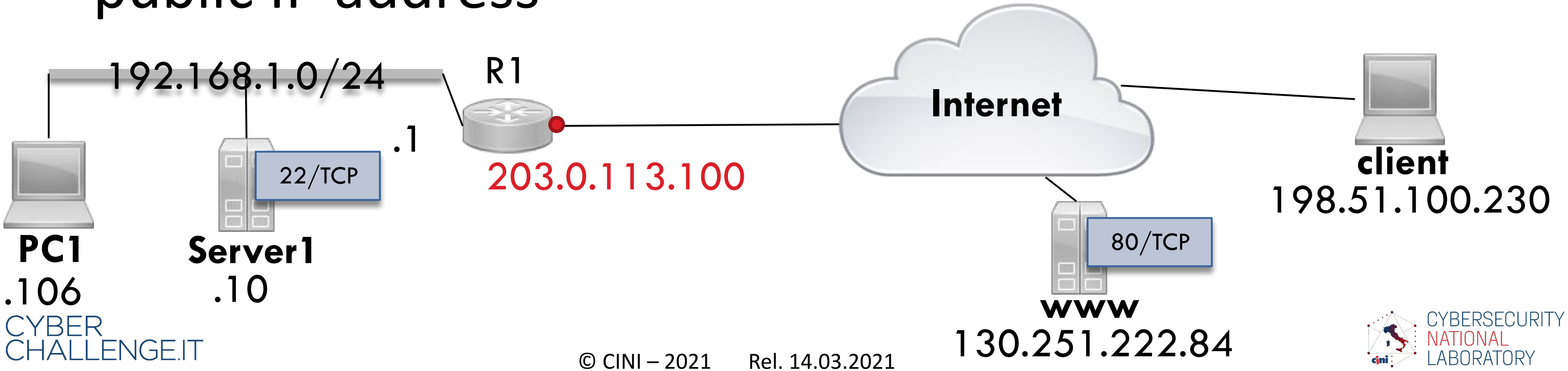
Network Address Translation

- Network Address Translation (NAT) generally involves rewriting the source and/or destination addresses of IP packets as they pass through a router or firewall
  - 192.168.1.0/24 is a private network and it is <u>not routable</u> on the Internet



Source NAT and Masquerade

- Masquerade is a **source NAT** rule, i.e., it is related to the source address of a packet
- The popular usage of NAT Masquerade is to translate a private address range to a single public IP address



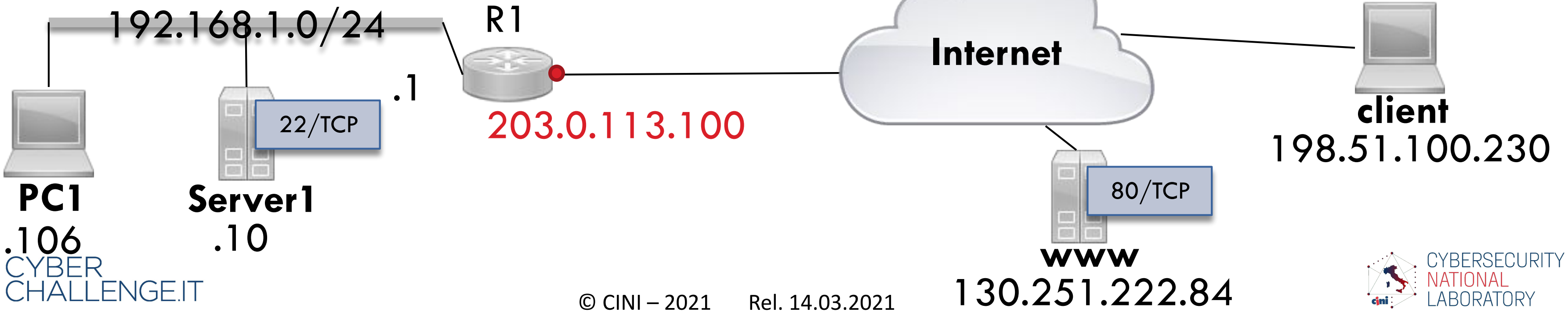
Source NAT and Masquerade (example)

- PC1 and Server1 accessing www (request)

SNAT table (dynamic)

203.0.113.100	52000,80	192.168.1.106
203.0.113.100	53000,80	192.168.1.10

SRCIP	SRCPORT	DSTIP	DSTPORT		SRCIP	SRCPORT	DSTIP	DSTPORT
192.168.1.106	52000	130.251.222.84	80	R1 →	203.0.113.100	52000	130.251.222.84	80
192.168.1.10	53000	130.251.222.84	80	R1 →	203.0.113.100	53000	130.251.222.84	80



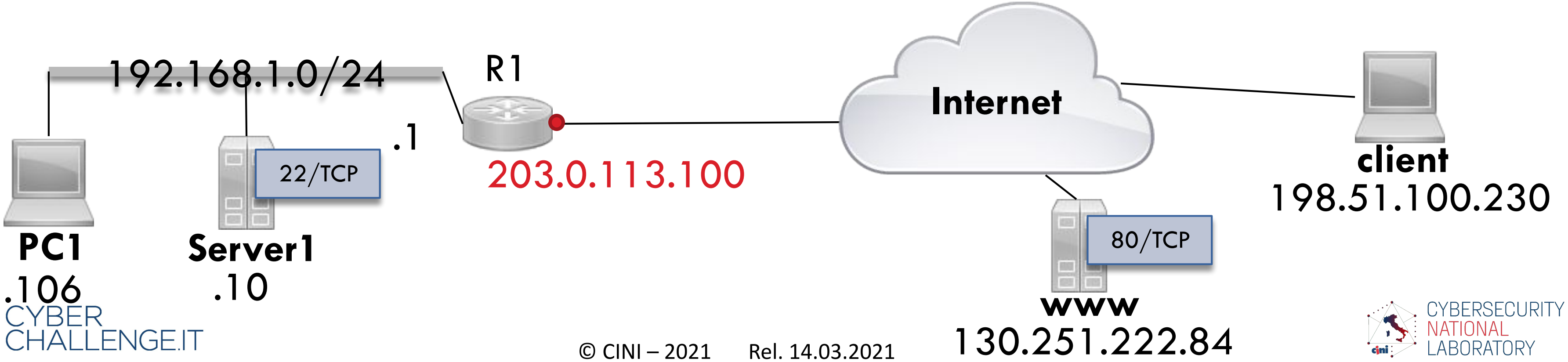
Source NAT and Masquerade (example)

- PC1 and Server1 accessing www (response)

SNAT table (dynamic)

203.0.113.100	52000,80	192.168.1.106
203.0.113.100	53000,80	192.168.1.10

SRCIP	SRCPORT	DSTIP	DSTPORT		SRCIP	SRCPORT	DSTIP	DSTPORT
130.251.222.84	80	192.168.1.106	5200	R1 ←	130.251.222.84	80	203.0.113.100	5200
130.251.222.84	80	192.168.1.10	5300	R1 ←	130.251.222.84	80	203.0.113.100	5300



Hypertext Transfer Protocol (HTTP)

- HTTP is a protocol which allows the fetching of resources, such as HTML documents
- HTTP is a client-server protocol
  - Requests are sent by one entity, namely the user-agent (e.g., a Web browser)
  - On the opposite side of the communication channel, is the server, which provides the document as requested by the client
  - A HTTP server uses the well-known port 80 TCP

HTTP messages

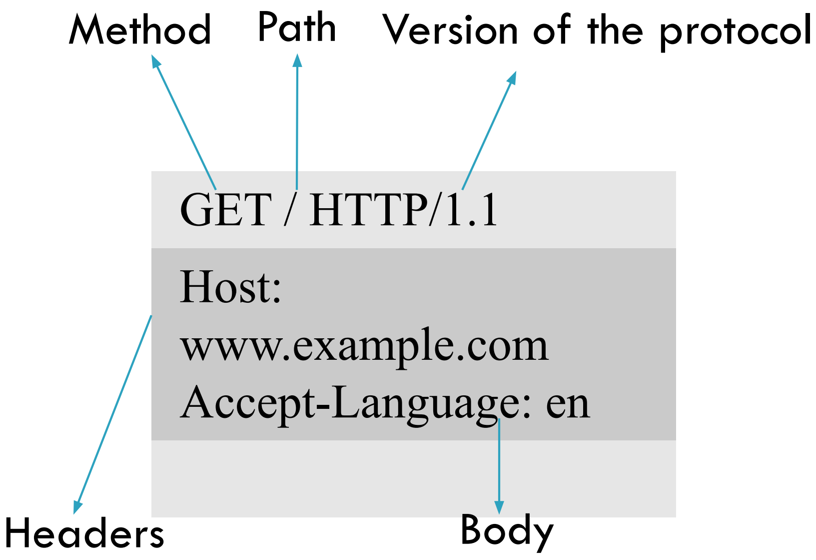
- Client and server exchange HTTP messages.
  - **HTTP Requests**: sent by the client to trigger an action on the server.
  - **HTTP Responses**: the answer from the server.
- HTTP messages are plain text, i.e., line-oriented sequences of characters.

Uniform Resource Locators (URLs)

- URL is the mechanism used by browsers to retrieve any published resource on the web

`http://`	`www.example.com`	`:80`	`/path/to/myfile.html`	`?key1=value1&key2=value2`	`#SomewhereInTheDoc`
the protocol to be used	the name of the web server	the port (usually omitted if it is the well-known)	the path to the resource on the web server.	extra parameters provided to the web server	fragment identifier: refers to a specific location within the resource being returned.

HTTP messages: requests



- **Method** defines the operation the client wants to perform. Typically, a client wants to fetch a resource (GET) or post the value of an HTML form (POST), though more operations may be needed in other cases
- **Path** corresponds to the URL of the resource stripped from elements that are obvious from the context (i.e., protocol, port, and domain)
- **Headers** (optional) convey additional information for the servers
- **Body** (optional): for some methods (e.g., POST) contains the resource sent

HTTP messages: responses

Status code Status message

```
HTTP/1.1 200 OK
Date: Fri, 29 Jan 2021 20:35:57 GMT
Server: Apache
Content-Length: 225
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE html...
```

Body Headers

- **Status code** indicates if the request was successful, or not, and why
- **Status message** is a non-authoritative short description of the status code
- **Headers** are like those for requests
- **Body** (optional) contains the fetched resource

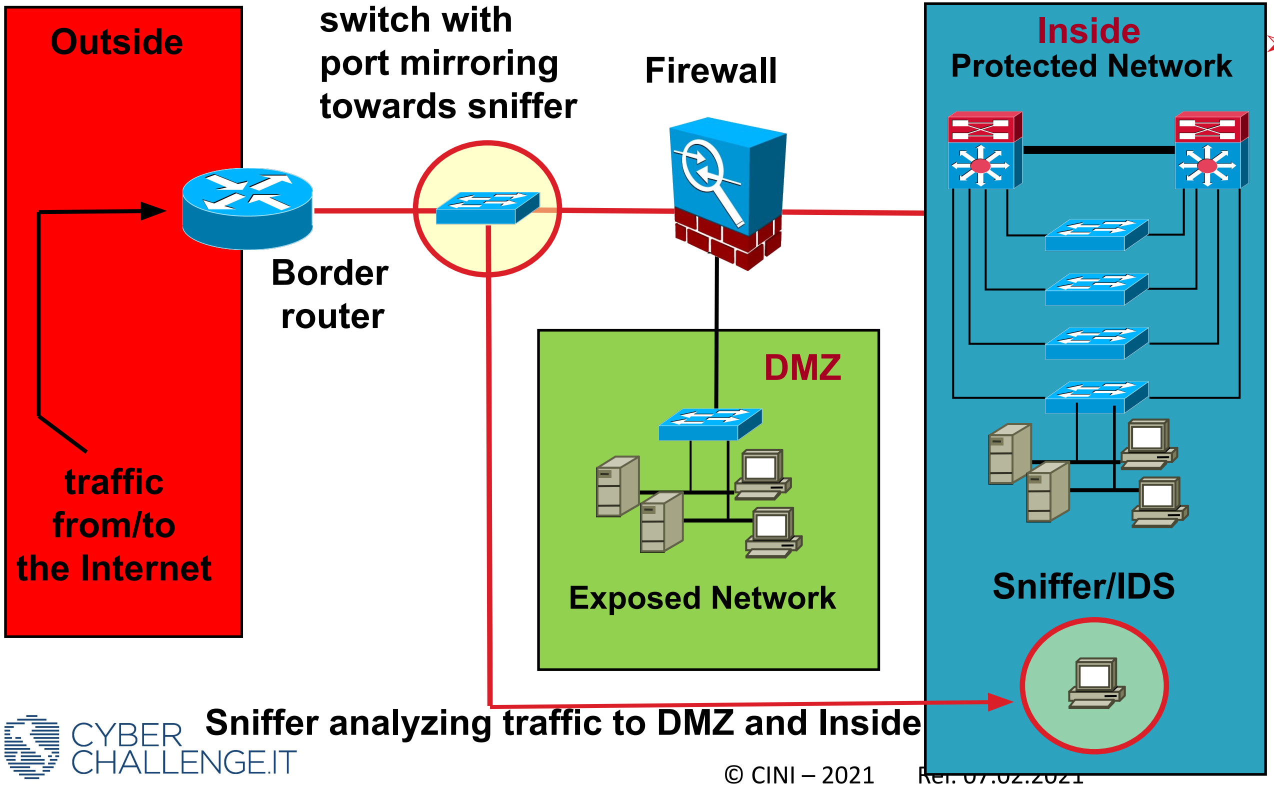
Francesco PALMIERI
Università di Salerno

Network analysis & monitoring tools

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

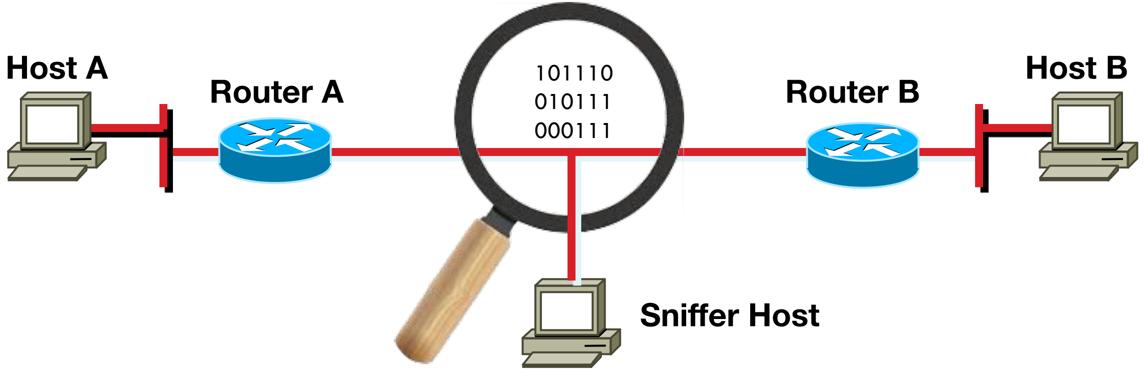
Basic security architecture



- In a common network architecture there are at least three domains:
  - **Outside** (all the world outside - the Internet): trust degree 0
  - **Inside** (the internal organization to be protected and hidden): degree of trust 100
  - **DMZ** (the set of internal machines that expose services outside): degree of trust 0 <x <100

Watching Traffic: Sniffing

- A sniffer is a software application that is capable of acquiring packets at the datalink level
- It is able to interpret clear information relating to level 2, 3 and 4 packet headers as well as application level protocols such as: FTP, HTTP, etc.
- A network adapter (NIC / TAP) programmed ad hoc (promiscuous mode) reads all packets in transit



Sniffing Applications

- **Automatic network analysis:** searching for specific patterns e.g., clear passwords and usernames: this is a common use for hackers / crackers;
- **Anomaly analysis**: in order to find out any problems within the networks, such as, why computer A cannot communicate with computer B;
- **Performance analysis**: to discover problems or bottlenecks in networks;
- **Detection of network intrusions**: to detect attacks or threats, as well as malicious activities in progress;
- **Recording of network traffic**: to create logs of network transactions available for subsequent "post-mortem" analysis.

Tcpdump: a simple CLI-based sniffer

<u>Sniffer</u>: Software or hardware tool that by telling on promiscuous mode configuration captures and allows the analysis of all the packages that pass through a network segment

<u>`tcpdump`</u> : Sniffer public domain based on Berkeley packet filter (BPF)

Available for download: `ftp://ftp.ee.lbl.gov/tcpdump.tar.Z`

```
23:06:37 10.1.101.1 > 224.0.0.10: ip-proto-88 40 [tos 0xc0]
```

time	source IP	dest IP	protocol	bytes	type of srv

Tcpdump: a simple CLI-based sniffer

```
08:08:16.155 spoofed.target.net.7 > 172.31.203.17.chargen: udp
```

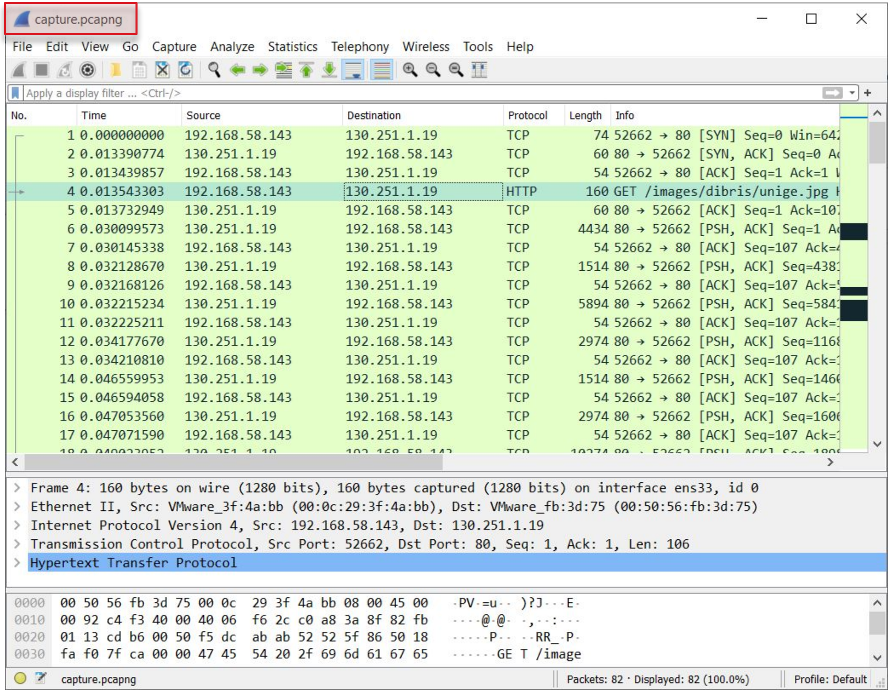
timestamp	src IP	src port	dst IP	dst port	protocol

- ➢ hosts can be referenced by name or IP address
- ➢ the ports can be specified by number or name of the service
- ➢ to specify a range of values, specific bytes must be pointed to

Wireshark

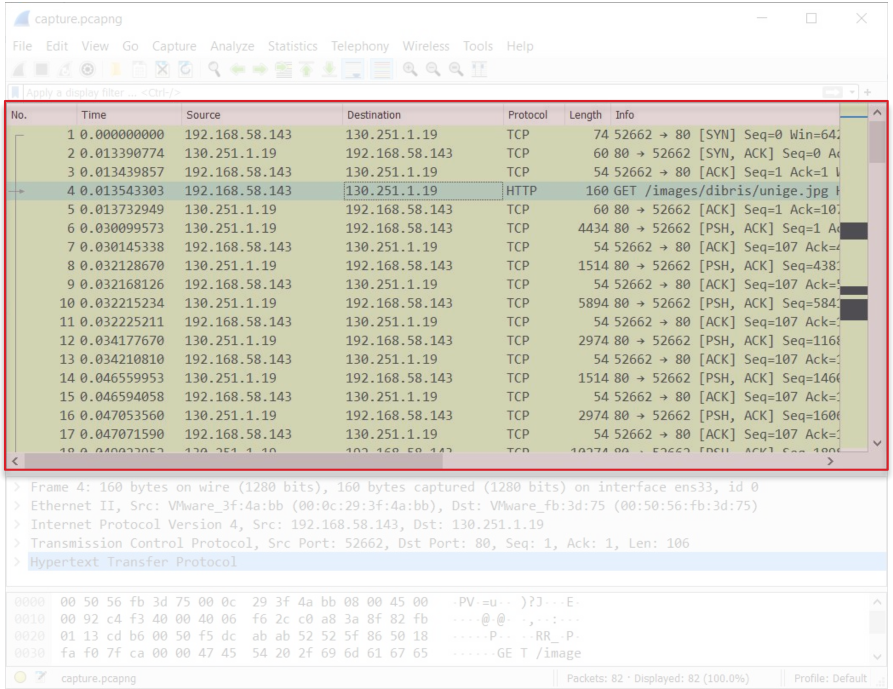
- Wireshark is a tool to capture data from a network (sniffer) and to analyse them
  - Analysis can be performed in real-time or on previously-recorded traffic files, through, e.g., *packet* capture or PCAP
  - Packets represent *generic* chunks of data and, depending on the considered level, can be interpreted as frames, datagram, or segment
- Available for UNIX and Windows: https://www.wireshark.org/

Wireshark GUI



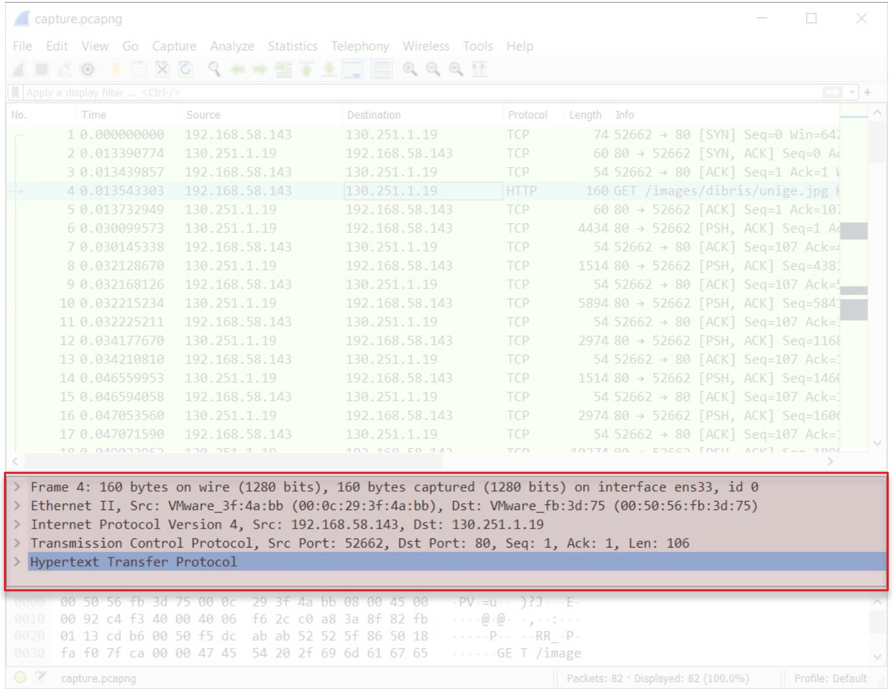
- Wireshark provides a Graphical User Interface (GUI)
- We detail its main elements as it appears after opening an existing PCAP file
  - From the File menu of the Start screen, use the command Open (CTRL-o) and select the PCAP file (e.g., capture.pcapng) to analyze

Wireshark GUI: packet list



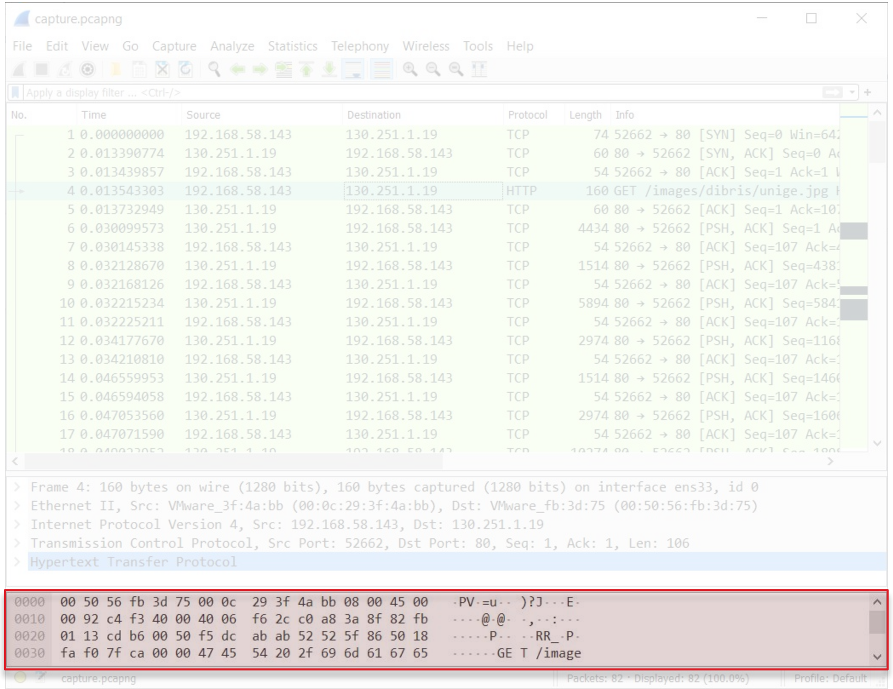
- The **packet list** pane displays a summary of each captured packet
- Each line in the packet list corresponds to one packet in the capture file (selecting a line in this pane displays more details in the *packet details* and *packet bytes* panes)
- Columns provide an overview of the packet
- You can click the column headings to sort by that value

Wireshark GUI: packet details



- The **packet details** pane shows the current packet (selected in the packet list pane) in a more detailed form
- In particular, it shows the protocols and fields of the packet in a tree, which can be expanded and collapsed

Wireshark GUI: packet bytes



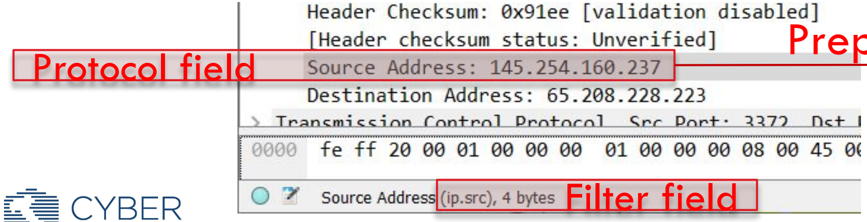
- The **packet bytes** pane shows the data of the current packet (selected in the packet list pane) in a hexdump style
- Each line contains
  - the data offset
  - sixteen hexadecimal bytes
  - sixteen ASCII bytes (Non-printable bytes are replaced with a period “.”)

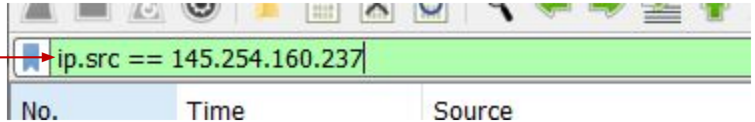
Display filters: filtering packets

- Wireshark provides a display filter language that enables you to precisely control which packets are displayed
- They can be used to check for
  - the presence of a protocol or field
  - the value of a field
  - compare two fields to each other
- These comparisons can be combined with logical operators and parentheses into complex expressions

Building filter expressions

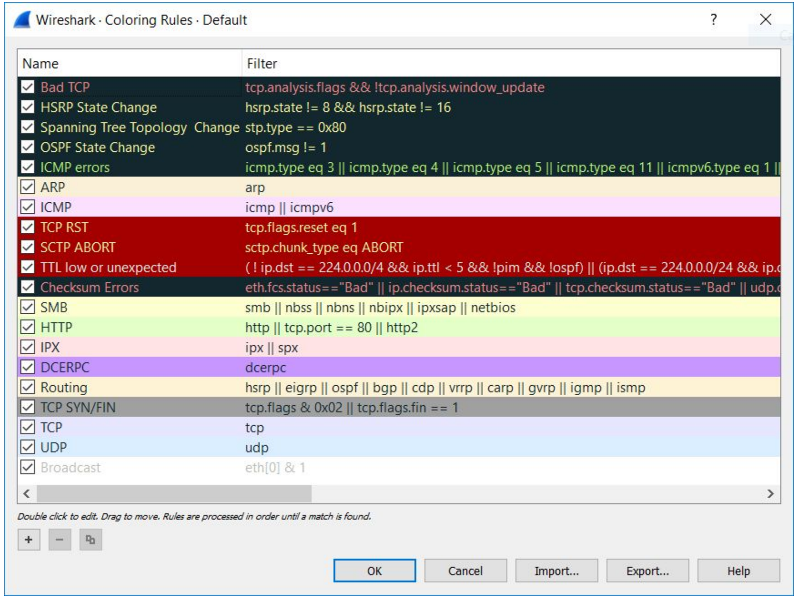
1. Help → Manual Pages → Wireshark Filters
2. Expression builder: right click on the toolbar → Display Filter Expression...
3. Select a protocols field in the packet details and use context menu entries:
   - ➢ Apply as Filter: filter the packet list with the selected key/value as the filter expression
   - ➢ Prepare a Filter: use the selected field key/value in the filter expression (filtering is not applied)





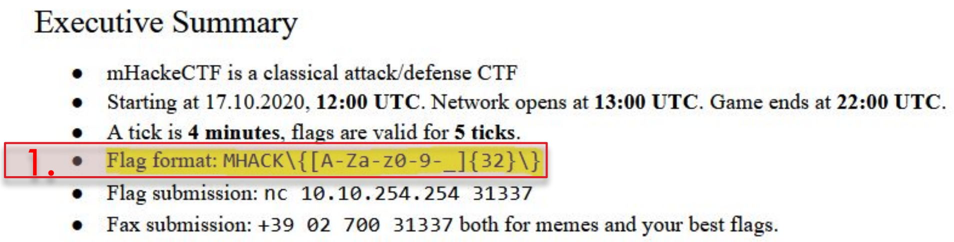
Coloring rules

- Wireshark supports coloring rules for packets
- View → Coloring Rules...

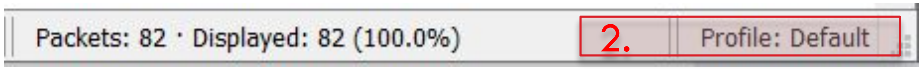


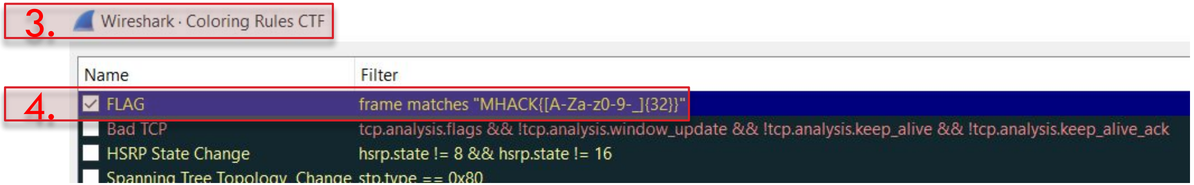
Coloring rules: A/D CTF example

1. Get the flag format from CTF rules
2. Right click on the Profile label of the Status Bar → New (e.g, CTF)
3. View → Coloring Rules… and disable all existing rules.
4. Add a new rule for highlighting flags

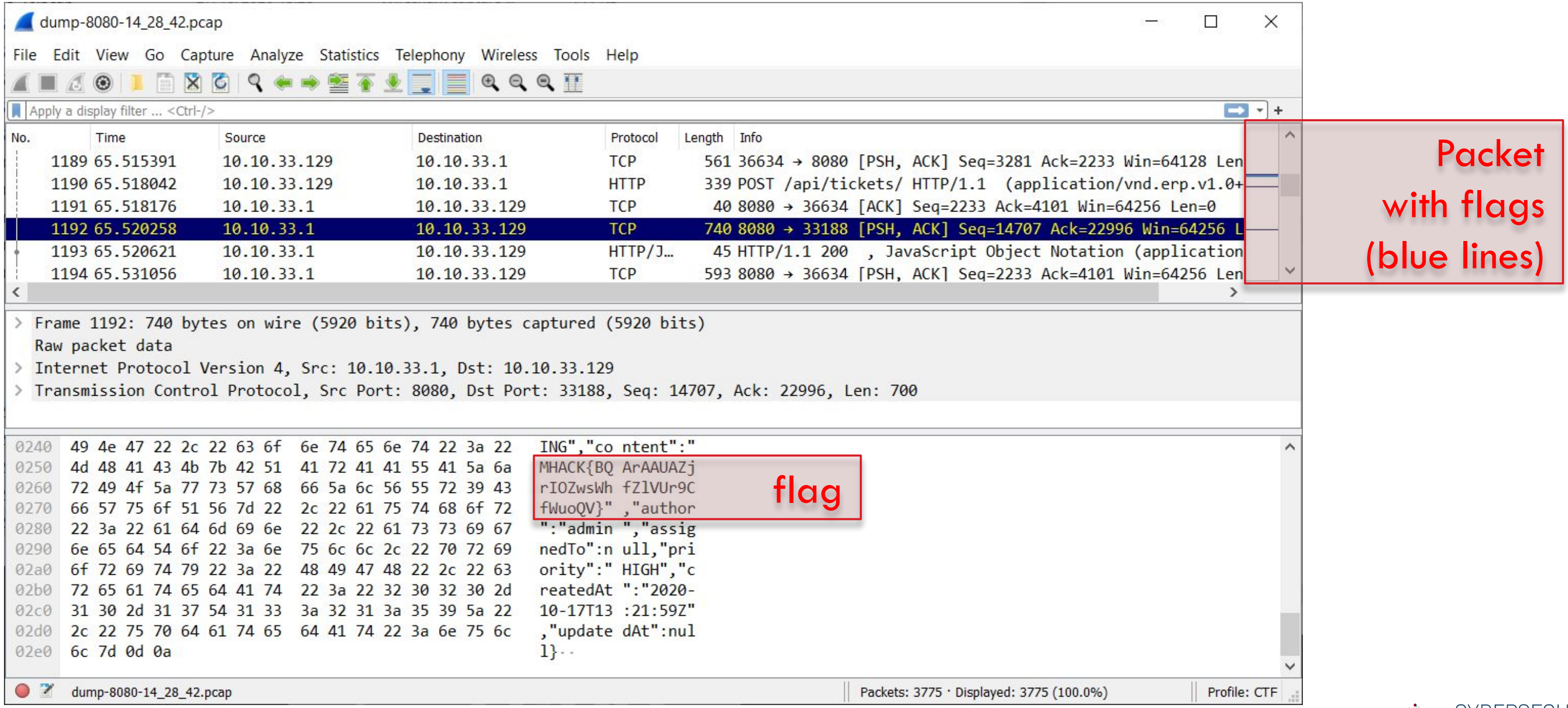

Executive Summary

- mHackeCTF is a classical attack/defense CTF
- Starting at 17.10.2020, **12:00 UTC**. Network opens at **13:00 UTC**. Game ends at **22:00 UTC**.
- A tick is **4 minutes**, flags are valid for **5 ticks**.
- Flag format: `MHACK\{[A-Za-z0-9-_]{32}\}`
- Flag submission: `nc 10.10.254.254 31337`
- Fax submission: `+39 02 700 31337` both for memes and your best flags.





Coloring rules: A/D CTF example

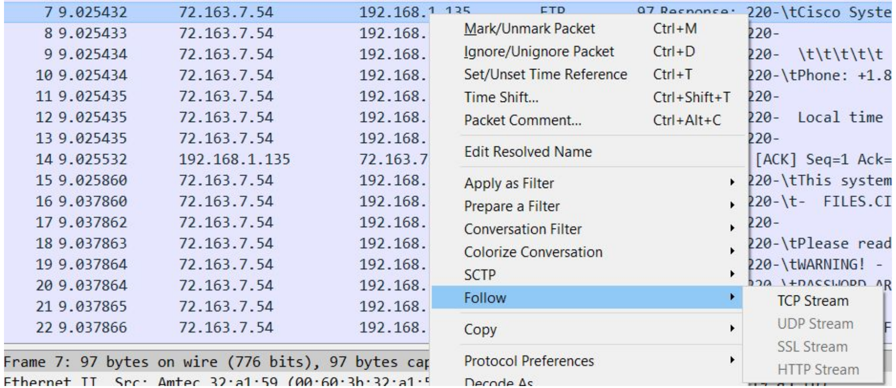


CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

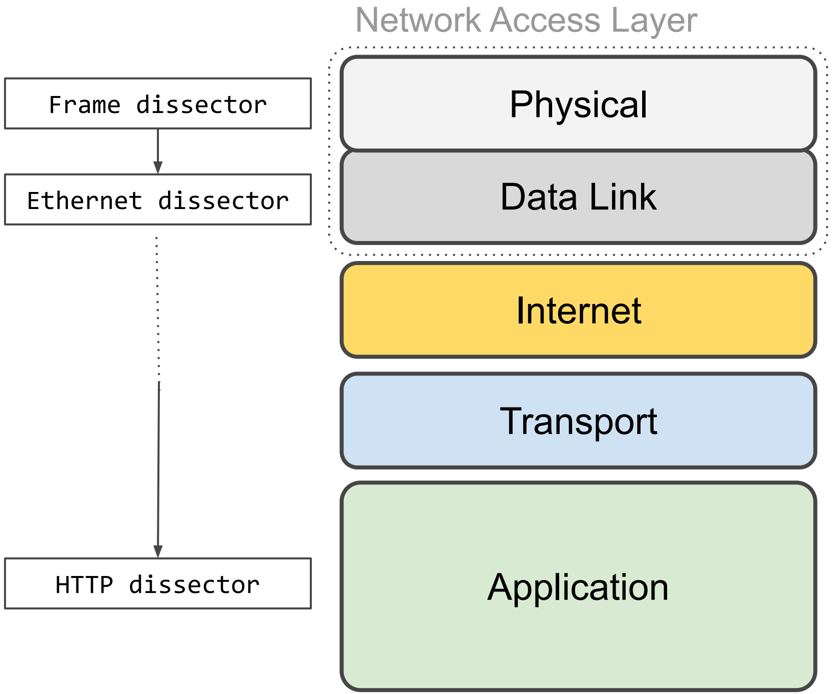
Follow streams

- **Follow stream** provides a different view on network traffic
- Instead of individual packets, one can see data flowing between client and server
- It can be enabled using the context menu in the packet list: a *display filter* which selects all the packets in the current stream is applied



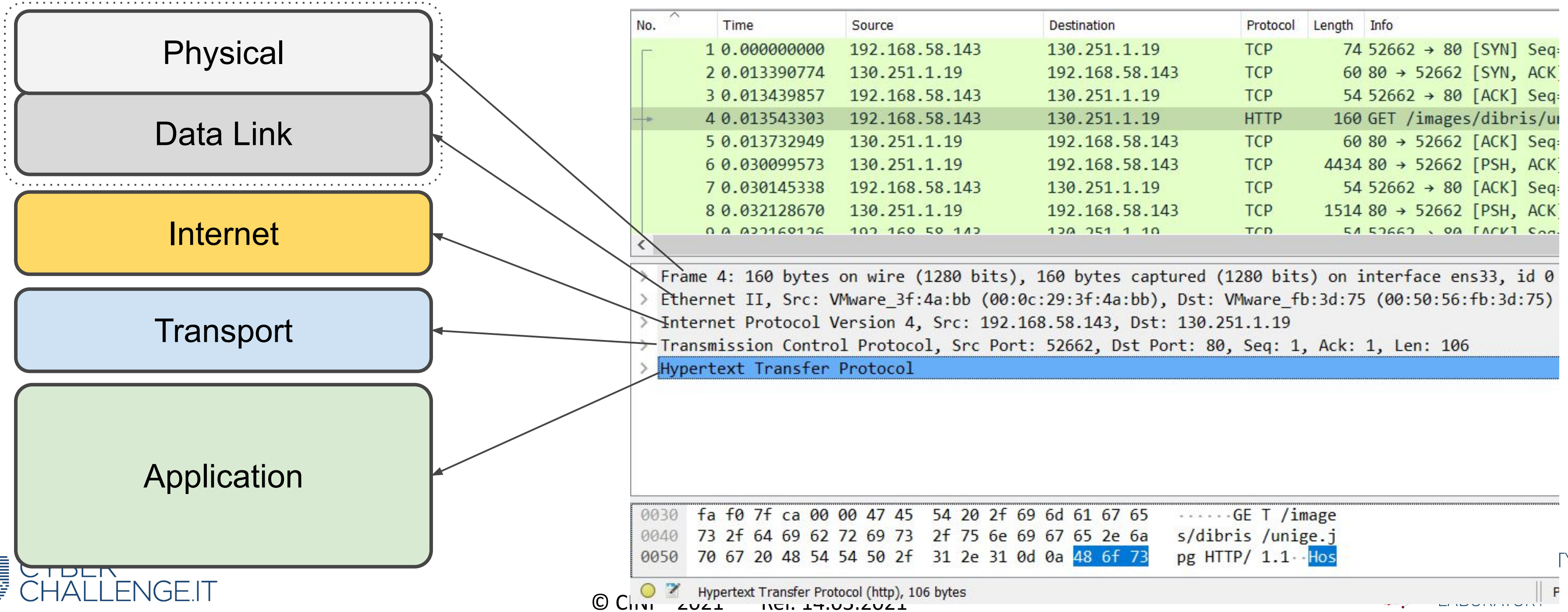
Dissectors

- **Dissectors** are (a kind of) plugin meant to analyze some part of a packet's data
- **Each protocol has its own dissector**, so dissecting a complete packet will typically involve several dissectors
- Find the right dissector to start decoding the packet data



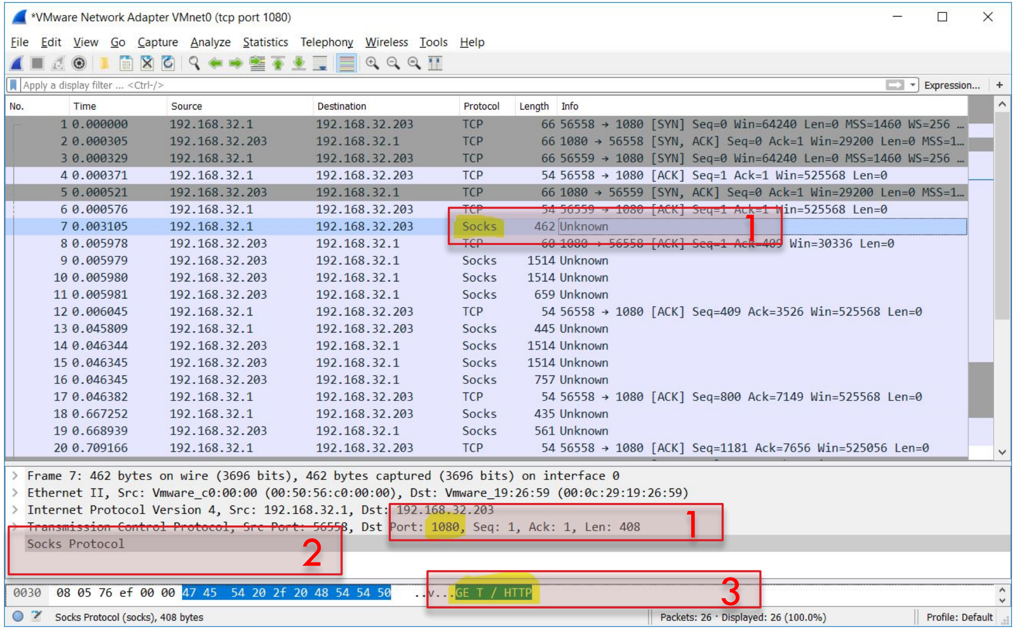
Packet details pane: dissectors

- The packet details pane shows outputs from the applied dissectors



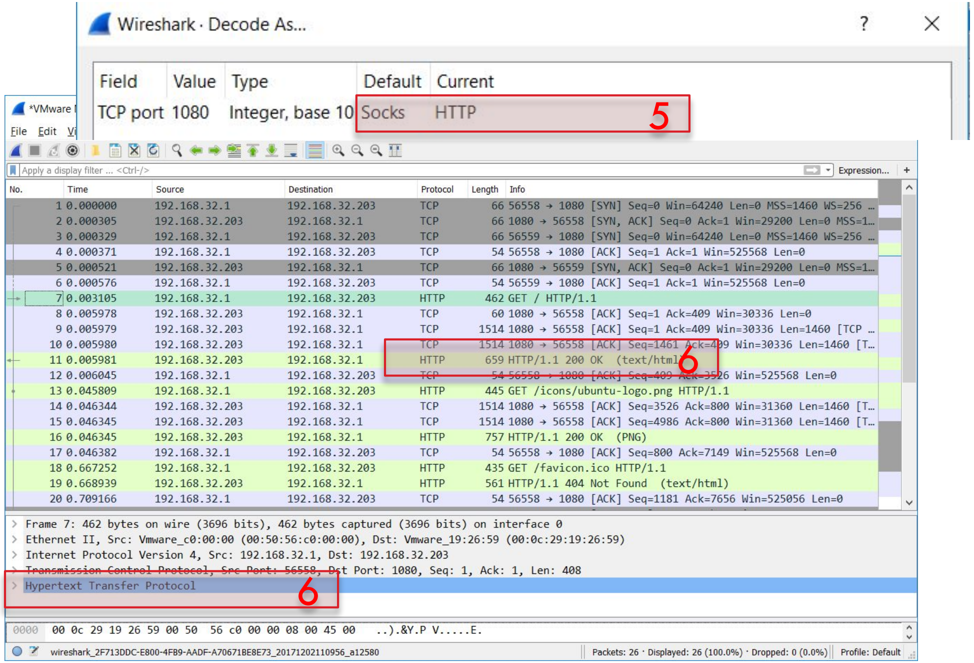
Change dissection rules (example)

1. Wireshark applies a Socks dissector, as the well-known port for Socks traffic is 1080/tcp
2. The dissector is not able to decode the data correctly (fields are empty in the packet details pane)
3. Raw data contain a GET / HTTP request string.



Change dissection rules (example)

4. Right click on (one of) the interested packet → Decode As…
5. Change the Current value (Socks) with the right dissector (HTTP)
6. Now protocol fields can be expanded in the packet details pane and visualized on the columns



Tshark

- TShark is a terminal oriented version of Wireshark
- Designed for capturing and displaying packets
- It supports the same options as Wireshark
  - *tshark –h*: print version and options
  - *man tshark*: linux manual
  - *online*: https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html

Tshark: examples

- Read PCAP files
  - `tshark -r <filename>`
- Detail output for specific protocols (available protocols: tshark –G protocols)
  - `tshark -O <protocol1>,<protocol2> -r <filename>`
- Filter output with a display filter (yank switch)
  - `tshark -Y <display_filter_expression> -r <filename>`
- Display specific protocols fields (available fields: tshark –G fields)
  - `tshark -r <filename>  -T fields -e <field1> … -e <fieldn>`
- Convert the hexadecimal payload into a binary files (data carving)
  - `tshark [… filtered data payload …] | xxd -r -p > <filename>`

PyShark

- Python wrapper for tshark, allowing python packet parsing using wireshark dissectors

- This package allows parsing from a capture file or a live capture, using all wireshark dissectors installed

Pyshark: examples

- Filtering packets by protocol
  - `filtered_cap = pyshark.FileCapture(path_to_file, display_filter='http')`
- Reading from a live interface
  - `capture = pyshark.RemoteCapture('192.168.1.101', 'eth0')`
- Access packet data (from destination IP)
  - `packet['ip'].dst`

    `192.168.0.1`
- Decrypting packet capture (from a PCAP file)
  - `cap1 = pyshark.FileCapture('/tmp/capture1.cap', decryption_key='password')`

Francesco PALMIERI
Università di Salerno

Network Security I

https://cybersecnatlab.it