

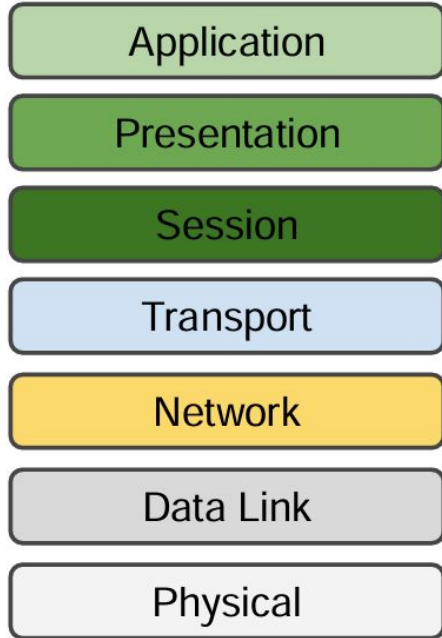
Introduction to Network & Automotive Security

m0leCon 2025 Workshops

Network fundamentals

- ISO/OSI model
 - TCP/IP model and protocols
 - Layer 2: Ethernet
 - Layer 3: Internet Protocol (IP)
 - Layer 4: UDP & TCP
 - Client-server model
-

ISO/OSI model



The Open Systems Interconnection (OSI) represents a guideline for network protocol design.

- Reference models for communication between different computers
- It is a theoretical model
- Uses a layered model (7 layers)
- Provides modularity and clear interfaces and allows extensibility
- Actual communication is defined by various protocols

ISO/OSI model

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 from source to destination

Data Link

It combines bits into a structure of data and provides their error-free transfer

Physical

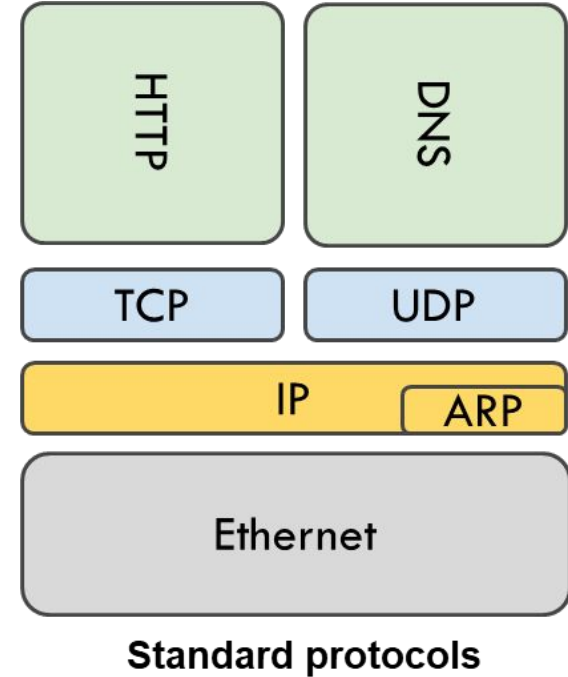
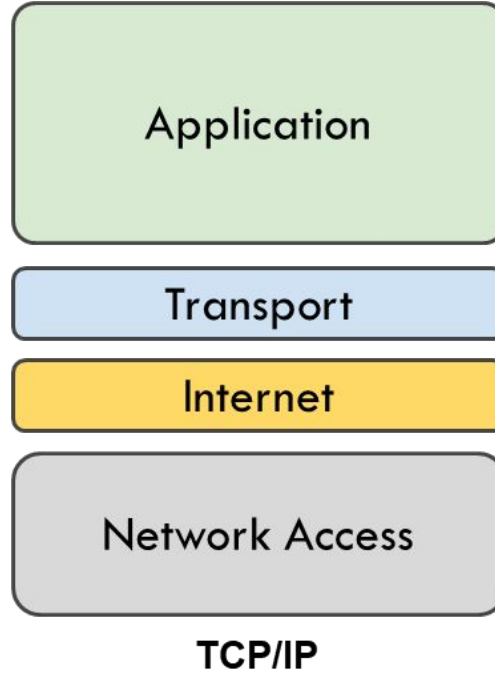
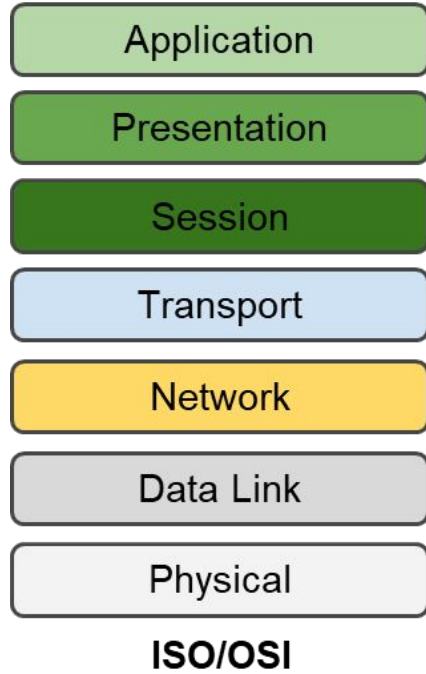
It provides a physical medium through which bits are transmitted

TCP/IP model

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:
 - 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



Layer 2: 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
 - 48 bits represented as 6 hexadecimal couples separated by colon: 01:02:03:aa:bb:cc (first 3-bytes: the ID of the manufacturer)
 - Every Ethernet device 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

Layer 3: Internet Protocol (IP)

- The standard for routing packets across interconnected networks
- Encapsulate data and pass that data in the form of packets
- Every device in the network is assigned a numerical label called IP address
- Two versions: IPv4 and IPv6
 - IPv6 is the “new” version that is being deployed to fulfill the need for more Internet addresses.
 - IPv4 currently still the most widely used
- Each packet includes:
 - The IP address of the source
 - The IP address of the destination

Layer 3: IP(v4) addressing

- 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)
- In every network, two addresses are used for special purposes:
 - 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
192 . 168 . 100 . 200

Layer 4: TCP & 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: 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)
 - Registered ports (1024-49151)
 - Ephemeral ports (49152–65535)
- The use of well-known and registered ports allows the requesting process to easily locate the corresponding server application processes on other hosts
- Despite these agreements, any service can listen on any port

Client-server model

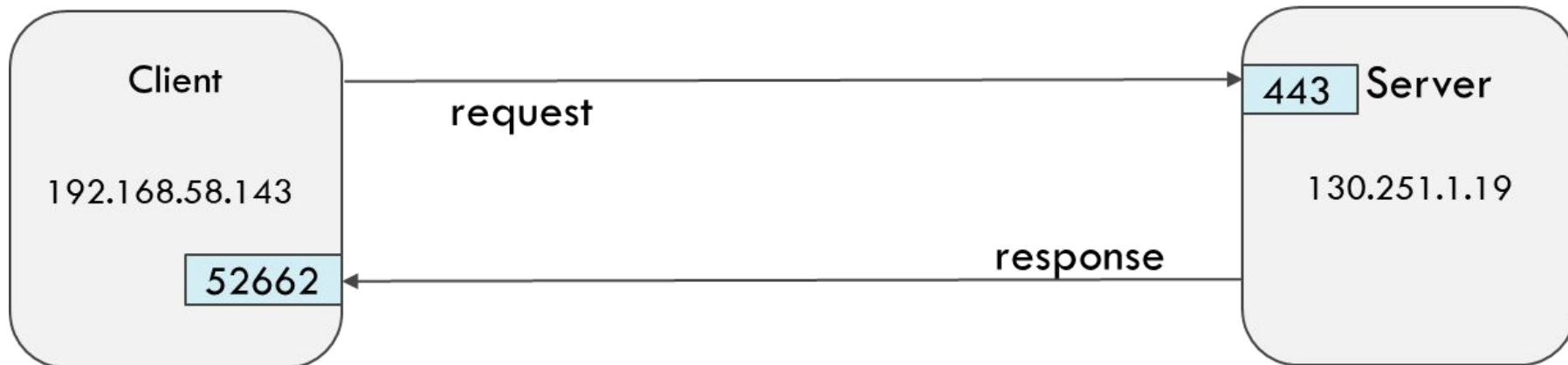
TCP/IP relies on the client-server model for enabling the process communication between network nodes.

- 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.

Client-server model (example)

192.168.58.143 52662 (ephemeral) 130.251.1.19 443 (well-known) https://...

Source IP	Source Port	Destination IP	Destination Port	Data
-----------	-------------	----------------	------------------	------

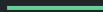


Data	Destination Port	Destination IP	Source Port	Source IP
------	------------------	----------------	-------------	-----------

52662 (ephemeral) 192.168.58.143 443 (well-known) 130.251.1.19

Network in CTF

- TCPDump
- Wireshark



Networking CTF challenges

In some CTF challenges, we are given a PCAP file.

Typically, solving these challenges requires analyzing the capture to find the flag by

- answering questions related to network traffic
- carving file from packet streams

TCPDump

CLI-based tool that allows the analysis and logging of all the packages that pass through a network segment

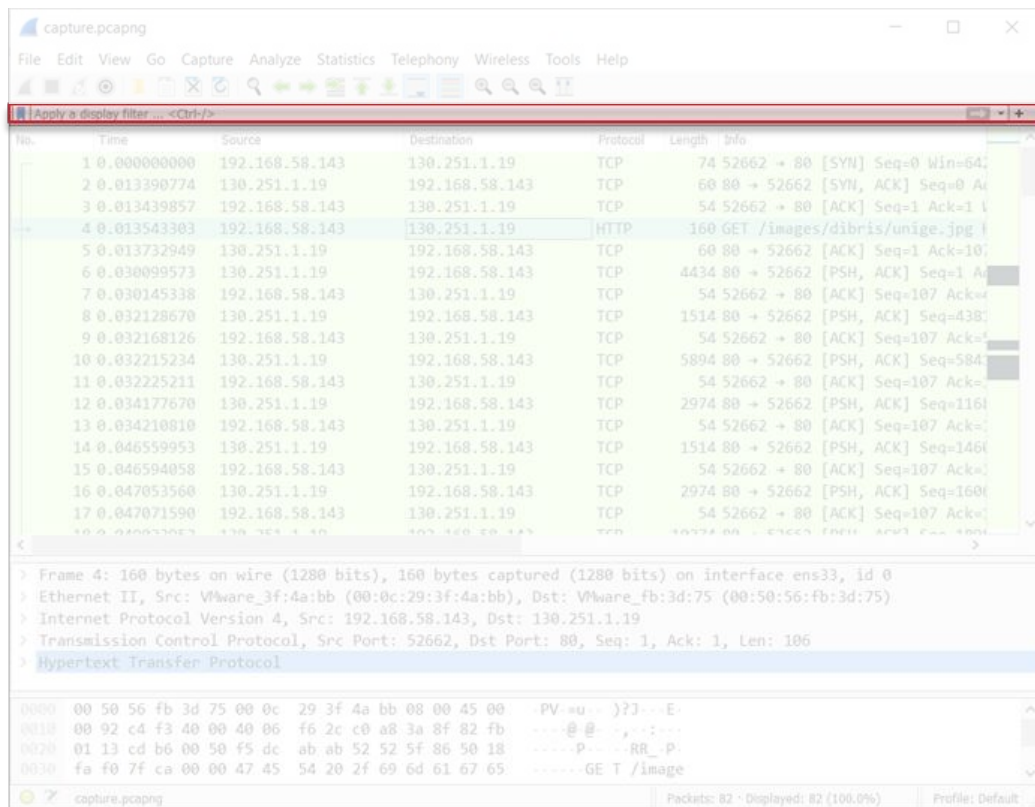
- Allows the filtering of captured packets through filtering expression
- The traffic gets saved in the packet capture file format with extension: .pcap

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/>

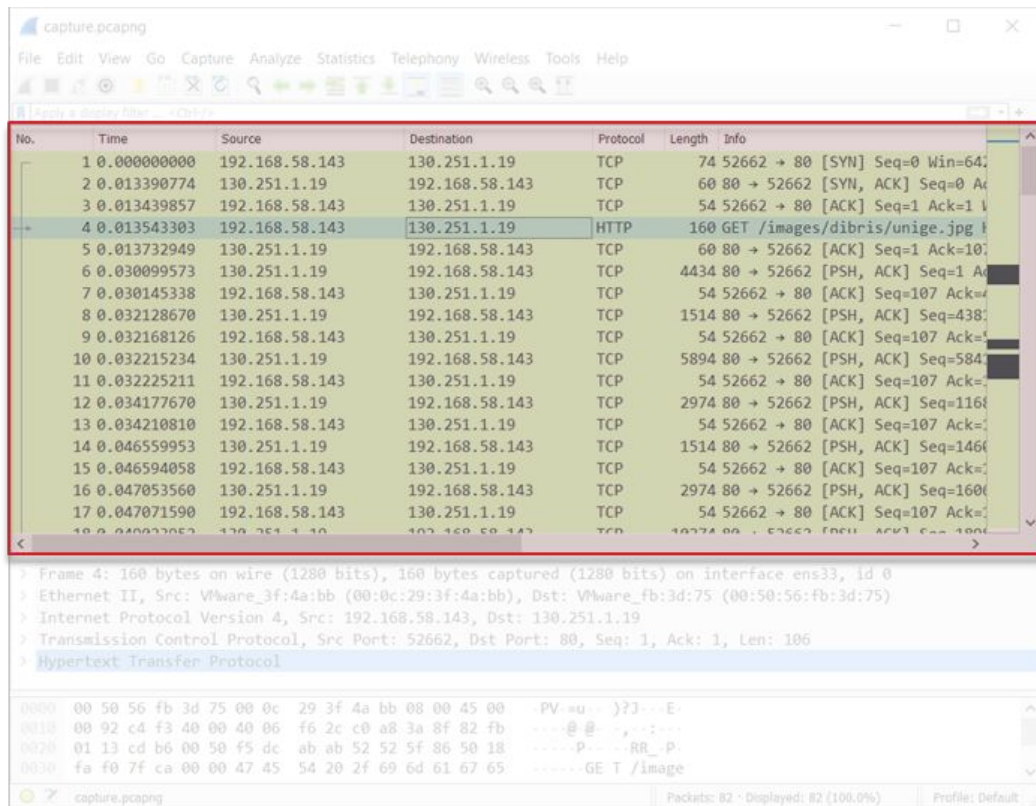
Wireshark GUI: filter toolbar



The filter toolbar lets you quickly edit and apply display filters

- Manage or select saved filters 
- Reset the current display filter and clear the edit area 
- Apply the current value in the edit area as the new display filter 

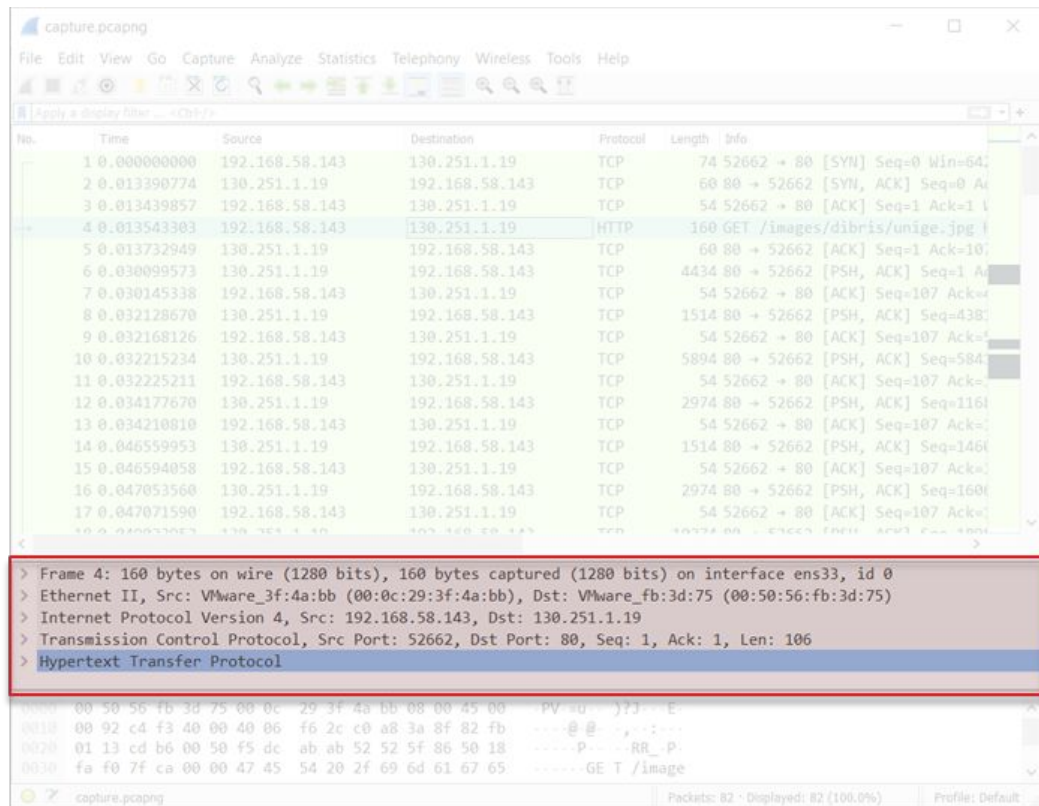
Wireshark GUI: packets list



The packets list pane displays a summary of each captured packet

- Each line in the packet list corresponds to one packet in the capture file
- 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

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

Useful filters

- `ip.src / ip.dst` → respectively filter by source and destination address
- To filter by protocol just use the name that appears in the protocol column, all in lowercase
- `protocol.port` → filter by port on the specified protocol
- `frame contains "string"` → filter all the packet that contain "string"
- `frame.len` → filter by packet length (size), in bytes
- Useful documentation links:
 - <https://wiki.wireshark.org/DisplayFilters>
 - <https://www.wireshark.org/docs/man-pages/wireshark-filter>

Follow streams (example)

- Telnet is a type of client-server protocol that can be used to open a command line on a remote host
- Blue is the data from the server to the client (e.g., the login: prompt)
- Red is the data from the client to the server (e.g., the user password is sent by the client and is not echoed by the server)
- Non-printable characters are replaced by dots.

```
.....!..".'.#...%.%.....!..".'.....P.....b.....b.... 8.
.....".'.#...&..$..&..$..#.....'.9600,9600....#.bam.zing.org:
0.0.....'.DISPLAY.bam.zing.org:0.0.....xterm-color.....!.....
OpenBSD/i386 (oof) (tty1)

login: .."....."ffaakkee
Password:user

Last login: Thu Dec  2 21:32:59 on tty1 from bam.zing.org
Warning: no Kerberos tickets issued.
OpenBSD 2.6-beta (OOF) #4: Tue Oct 12 20:42:32 CDT 1999

Welcome to OpenBSD: The proactively secure Unix-like operating system.

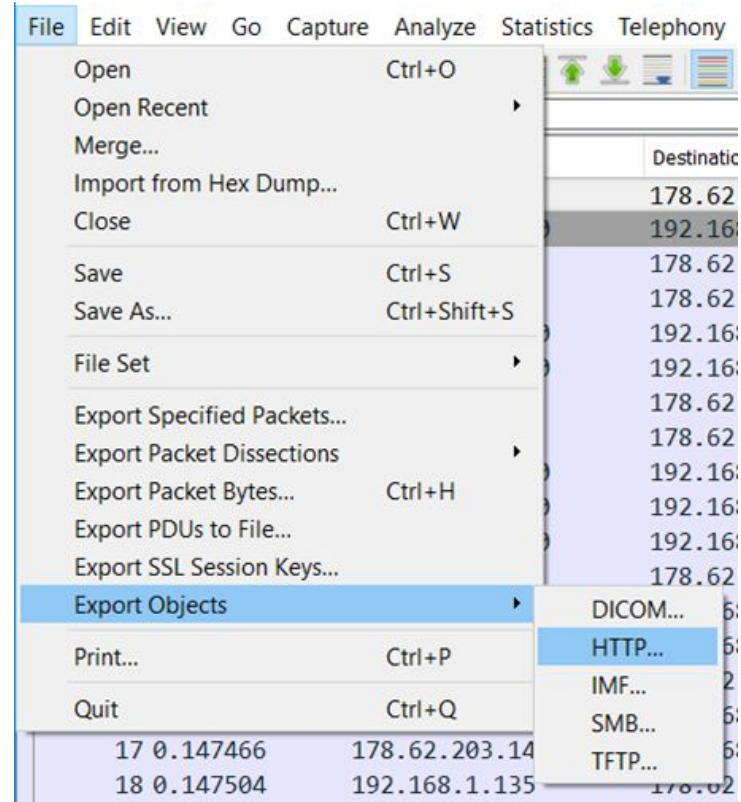
Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code. With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

$ llss
.
$ llss --aa
.
.      .cshrc  .login  .mailrc  .profile  .rhosts
$ //ssbbiinn//ppiinnngg  wwwwww..yyaahhoooo..cccoomm

PING www.yahoo.com (204.71.200.74): 56 data bytes
64 bytes from 204.71.200.74: icmp_seq=0 ttl=239 time=73.569 ms
64 bytes from 204.71.200.74: icmp_seq=1 ttl=239 time=71.099 ms
64 bytes from 204.71.200.74: icmp_seq=2 ttl=239 time=68.728 ms
64 bytes from 204.71.200.74: icmp_seq=3 ttl=239 time=73.122 ms
64 bytes from 204.71.200.74: icmp_seq=4 ttl=239 time=71.276 ms
64 bytes from 204.71.200.74: icmp_seq=5 ttl=239 time=75.831 ms
64 bytes from 204.71.200.74: icmp_seq=6 ttl=239 time=70.101 ms
64 bytes from 204.71.200.74: icmp_seq=7 ttl=239 time=74.528 ms
64 bytes from 204.71.200.74: icmp_seq=9 ttl=239 time=74.514 ms
64 bytes from 204.71.200.74: icmp_seq=10 ttl=239 time=75.188 ms
64 bytes from 204.71.200.74: icmp_seq=11 ttl=239 time=72.925 ms
...^C
---- www.yahoo.com ping statistics ----
13 packets transmitted, 11 packets received, 15% packet loss
round-trip min/avg/max = 68.728/72.807/75.831 ms
$ eexxiitt
.
```


Export objects (example)

- File → Export Objects
- This feature scans through (some) protocol streams and takes reassembled objects (e.g., HTML docs, images, executables)
- These files can be exported and saved to disk



Challenge time



Packet analysis and manipulation

- Tshark
- Pyshark
- Scapy

Tshark

- TShark is a terminal oriented version of Wireshark
- Designed for capturing and displaying packets
- It supports the same options as Wireshark

Pyshark

- It's a Tshark wrapper for python
- Useful to automate operations on large numbers of packets
- After loading the pcap it exposes a list containing all the packets parsed from the capture
- Packets are divided in layers. You need to access the correct layer before querying the wanted field
 - `packet.ip.dst` → access the IP layer and selects the destination IP field
 - `packet.tcp.payload` → access the TCP layer and selects the payload field
- To check if a layer is in a packet the layer name can be used: `if 'IP' in packet`
- To get all available fields, the attribute `packet.layer.field_names` can be used (eg: `packet.ip.field_names`)

Pyshark: example

Simple script to print all the TCP packets payloads from a pcap file

```
import pyshark

cap = pyshark.FileCapture('/path/to/pcap/file.pcap')

for packet in cap:
    # Check whether packet is TCP (check is case insensitive)
    if 'tcp' in packet:
        try:
            print(packet.tcp.payload.binary_value)
        except:
            continue
```

Scapy

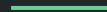
- Scapy is a powerful interactive packet manipulation library written in Python.
- It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more.

Challenge time



CAN

- General overview
- Protocol overview
- Tools
- Attacks
- Architecture of a modern vehicle



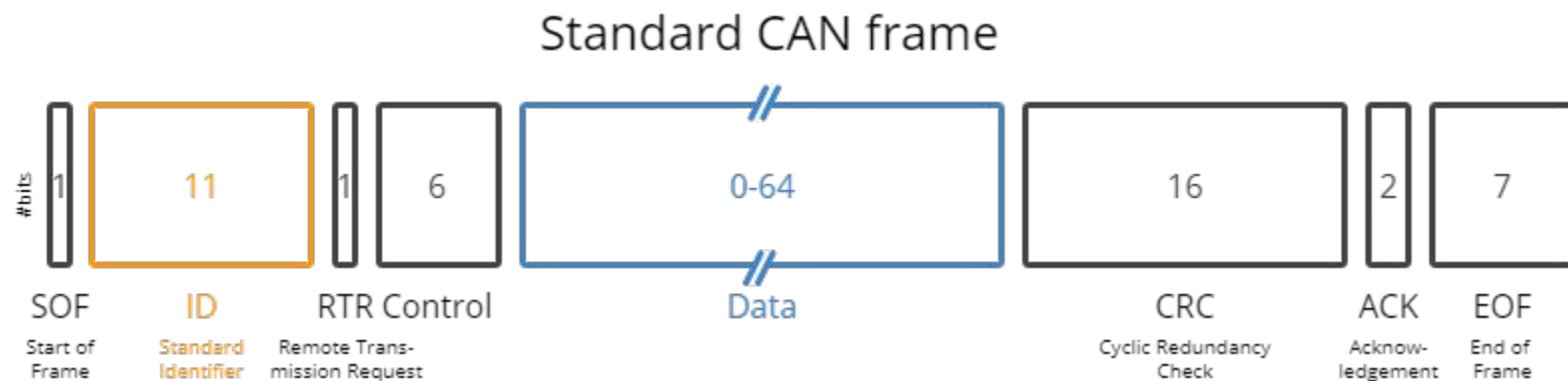
General overview

- Controller Area Network (CAN) is a broadcast digital serial bus designed to operate at speeds from 10kbit/s to 1Mbit/s
- It was introduced by Bosch in the early 1980s for automotive applications
- Flexible: nodes can easily be added or removed.
- Lower cost compared to ethernet

Protocol overview

- CAN protocol uses a broadcast transmission , every node (i.e., ECU) receives every message
- CAN is a message based protocol . A message is organized in a structure called frame
- The CAN frames do not contain addresses of either the transmitting node or any of the intended receiving node(s). An arbitration ID that is unique across the network labels the frame
- Depending on the arbitration ID of that transmitted frame, each CAN node decides whether to accept the frame, or not
- If multiple nodes try to transmit a message at the same time, the node with the highest priority (lowest arbitration ID) automatically gets bus access

Protocol overview



Tools

- `candump` & `isotpdump`:
 - tools to view and log CAN packets over a can bus
- `cansend` & `cangen`:
 - tools to generate and send random and/or arbitrary can packets over a CAN bus
- `caring` `caribou` & `gallia`:
 - tools to test the security of a CAN bus and connected devices
- `Scapy` & `python-can`:
 - libraries to provide CAN bus support to python

Attack: DOS

Attack: DOS

Execution of the Attack:

The attacker sends rapid high-priority messages after gaining physical or remote access to the CAN bus.

Attack: DOS

Execution of the Attack:

The attacker sends rapid high-priority messages after gaining physical or remote access to the CAN bus.

Impact on the System:

Critical messages are blocked, disrupting essential vehicle functions like brakes or engine control, leading to unsafe operation.

Attack: Spoofing/replay

Attack: Spoofing/replay

Execution of the Attack:

The attacker intercepts valid CAN messages, then modifies or replays them to manipulate vehicle behavior, often requiring physical or remote access to the CAN bus.

Attack: Spoofing/replay

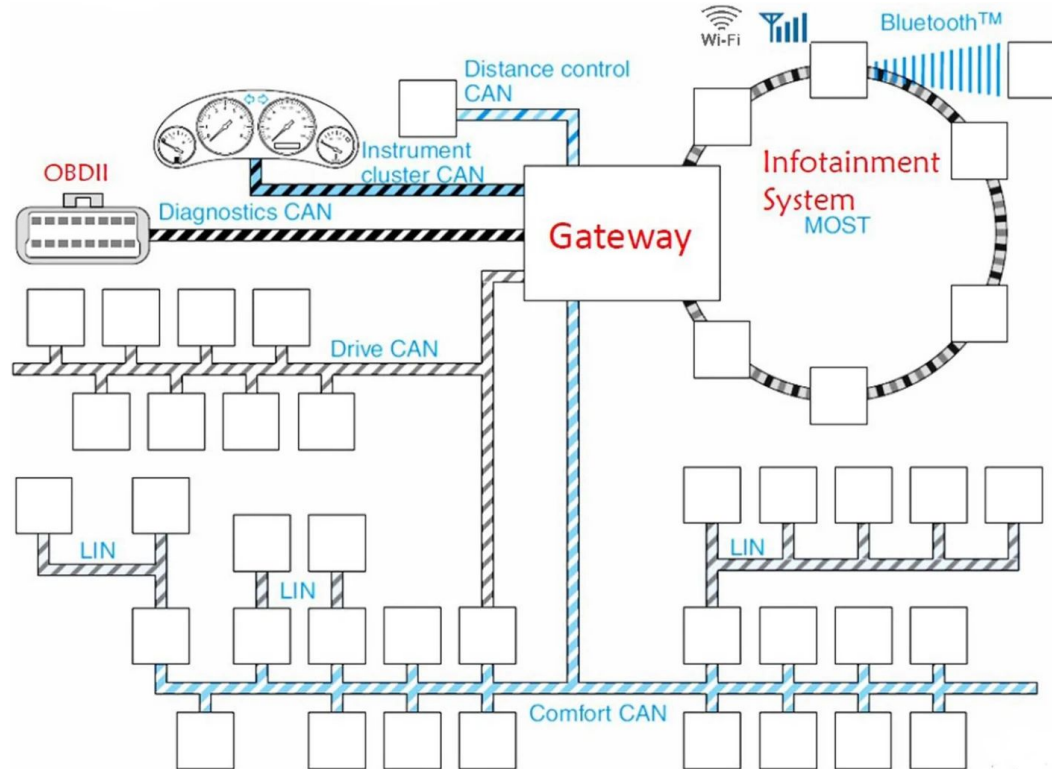
Execution of the Attack:

The attacker intercepts valid CAN messages, then modifies or replays them to manipulate vehicle behavior, often requiring physical or remote access to the CAN bus.

Impact on the System:

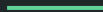
Maliciously injected or replayed messages can trigger unintended actions, such as unlocking doors, disabling safety features, or altering vehicle performance.

Architecture of a modern vehicle



UDS

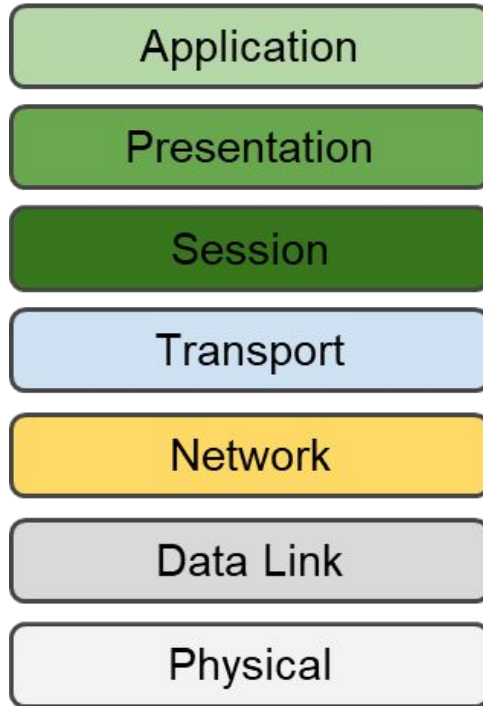
- General Overview
- Protocol Overview
- Attacks



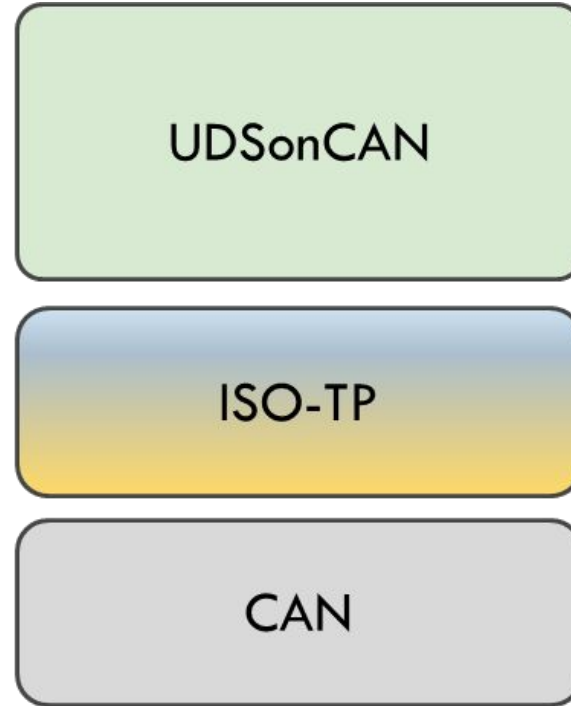
General overview

- Unified Diagnostic Services (UDS) is a communication protocol used in automotive Electronic Control Units (ECUs) to enable diagnostics, firmware updates, routine testing and more
- The UDS protocol builds on top of the CAN protocol on OSI layers 5 and 7
- Communication is performed in a client-server relationship - with the client being a tester-tool and the server being a vehicle ECU

General overview



ISO/OSI



UDSonCAN

Protocol overview: sessions

At the base of the protocol are sessions and services

A session is a mode of operation that determines the type of diagnostics the ECU (Electronic Control Unit) will allow. Each session has its own set of permissions and capabilities. The most common UDS sessions are:

- Default Session (0x01): normal operating mode, limited diag functionality
- Programming Session (0x02): enables reprogramming or flashing the ECU firmware, allows access to memory regions and advance functions
- Extended Session (0x03): advanced diagnostics, access to tests and parameter adjustments

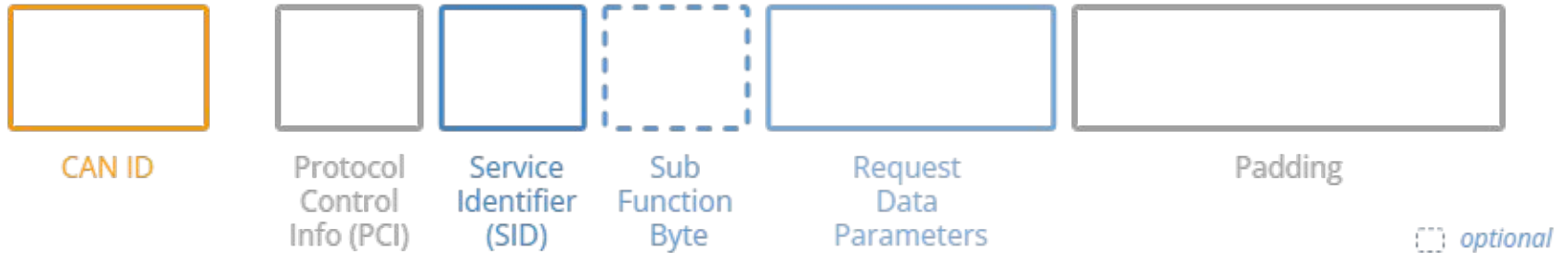
Protocol overview: services

Services are standardized commands used to interact with ECUs. Each service is identified by a unique service identifier (SID), usually a single byte. The structure of a UDS request typically includes the SID and optional parameters. They allow to:

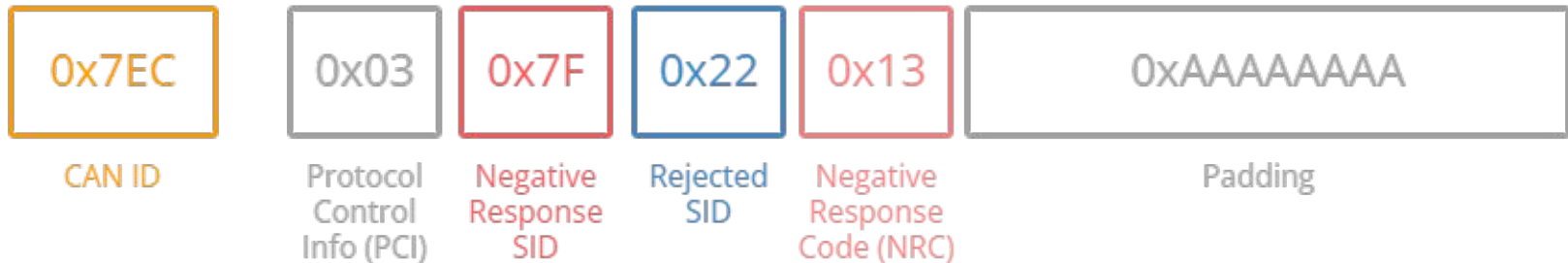
- Read diagnostic informations
- Read and write ECU memory
- Instruct the ECU to start routines
- Perform client authentication and authorization

Protocol overview

UDS request message structure (UDS on CAN)



UDS Negative Response example (UDS on CAN)



Attack: Exploiting weak auth algorithm

Attack: Exploiting weak auth algorithm

Execution of the Attack:

Exploiting weak authentication algorithms allows attackers to bypass security, enabling unauthorized access to diagnostic functions or sensitive vehicle data.

Attack: Exploiting weak auth algorithm

Execution of the Attack:

Exploiting weak authentication algorithms allows attackers to bypass security, enabling unauthorized access to diagnostic functions or sensitive vehicle data.

Impact on the System:

The attacker analyzes weak cryptographic mechanisms or default keys in the UDS protocol, then crafts malicious requests to gain control or retrieve confidential information.

Attack: Improper protection of critical services

Attack: Improper protection of critical services

Execution of the Attack:

The attacker exploits insufficient authentication or access control to misuse functions like **ReadDataByAddress** to extract sensitive information or **WriteDataByAddress** to modify critical system parameters or firmware.

Attack: Improper protection of critical services

Execution of the Attack:

The attacker exploits insufficient authentication or access control to misuse functions like **ReadDataByAddress** to extract sensitive information or **WriteDataByAddress** to modify critical system parameters or firmware.

Impact on the System:

Improper protection of critical services, such as **ReadDataByAddress** and **WriteDataByAddress**, can allow attackers to access sensitive data or alter firmware, leading to unauthorized control, system instability, or safety risks.

Challenge time

