Hands-on Car Hacking & Automotive Cybersecurity

Workshop by Roald Nefs

Agenda_

- Intro
- Icebreaker
- Background & Context
- Exercise 1
- Exercise 2
- Regroup
- Next Steps

About me_

Roald Nefs

Chief Technology Officer at **Warpnet**

Board Member at **Stichting Friendly Cyber**

Organizer at **BSides Groningen**, **BSides Amsterdam**, and **Cloud Native Groningen**

Hardware Hacker at home

Disclaimer_

This workshop is for **educational purposes only**.

Hacking cars:

- 🧱 **Brick your car** (permanently damage ECUs or components)
- ⚠️ **Modify safety-critical systems** (e.g. brakes, steering, airbags)
- 🚫 **Break the law** (e.g. tampering with odometer or VINs)

Proceed at your own risk. Only work on vehicles you own or have explicit permission to test.

Getting the PDF_

The PDF can be downloaded at:

https://roaldnefs.com/bsidesluxembourg.pdf

Threat Modeling modern connected cars_

EXERCISE

Threat Modeling_

🛠️ **Scenario:**
Imagine you are an attacker targeting a modern connected car. List as many attack surfaces or entry points as possible in 5 minutes.

Modern cars_

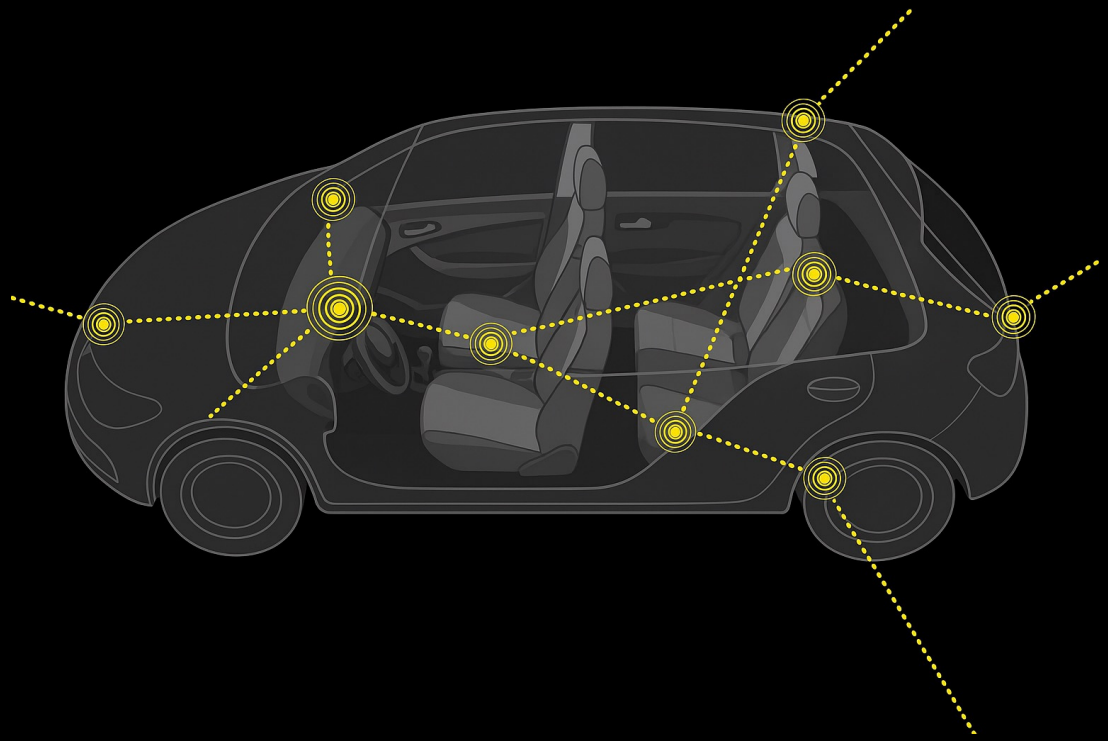
Modern cars are computers on wheels:

- **Driving assistance:** anti-collision, line detection, ESP/ABS, road sign reconnaissance
- **Connectivity:** GPS, LTE, WiFi, Bluetooth
- **Onboard services:** remote control, localization, auto-diagnostic
- **Emissions:** fuel efficiency, hybrid systems, battery optimization
- **Autonomous driving:** Honda reached level 3 autonomous driving in 2021

ECUs_

ECU: Electronic **C**ontrol **U**nit

- Acts as the brain of a car subsystem.
- Reads data from sensors, manage actuators (e.g. throttle, brakes, steering).
- Communicates with other ECUs, external devices, and backend servers of wired or wireless networks (e.g. CAN, Ethernet, Bluetooth, cellular)
- May be built using microcontrollers (MCUs) or more powerful systems-on-chip (SOC), depending on complexity.



Common ECUs_

ECU are commonly labeled using abbreviations, which may vary between manufacturers.

Common ECU names

- **ECU:** Engine Control Unit
- **TCM:** Transmission Control Module
- **PCM:** Powertrain Control Module
- **BCM:** Body Controle Module
- **ICM:** Instrument Cluster Module
- **ABS:** Anti-lock Braking System
- **ADAS:** Advanced Driver Assistance System
- **IVI:** In-Vehicle Infotainment
- **BMS:** Battery Management System

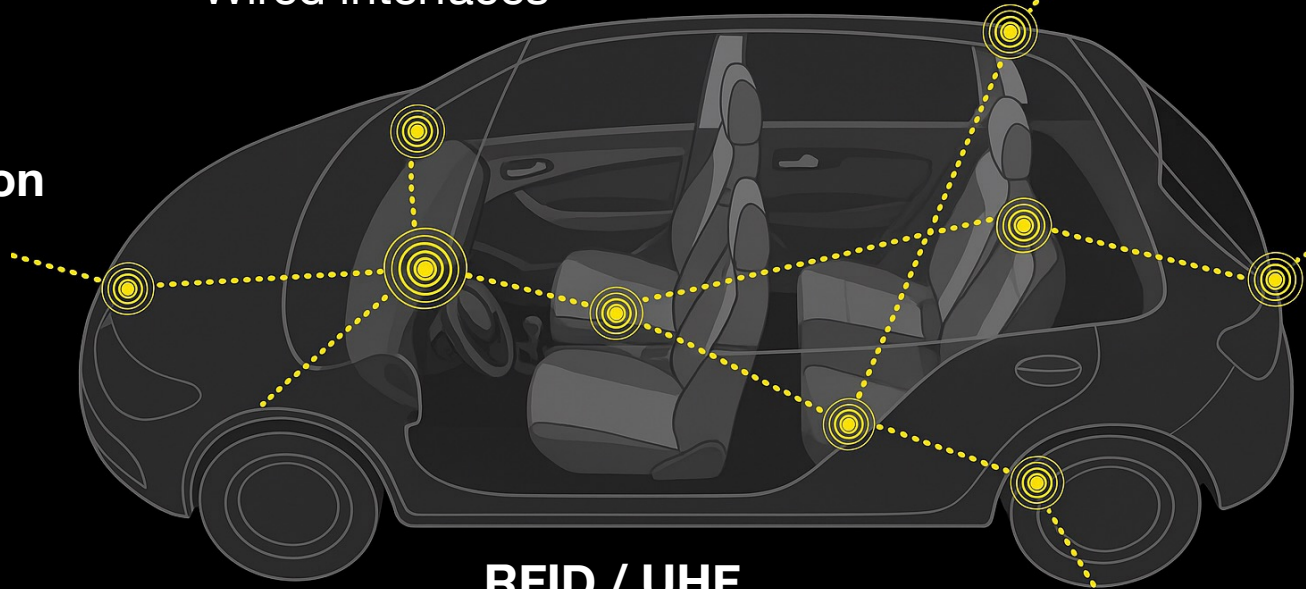
Connected ECUs_

ECU networks in modern cars. Different networks balance cost, speed, robustness, and safety depending on ECU needs. Some ECUs connect to multiple networks.

Key network types

- **CAN:** Controller Area Network, robust and widely used e.g. engine, brakes, airbags
- **LIN:** Local Interconnect Network, low-cost, used for window motors, mirrors
- **Automotive Ethernet:** high-speed, used for ADAS, cameras, infotainment
- **FlexRay:** high-speed, fault-tolerant, used advanced braking, steer-by-wire
- **MOST:** Media Oriented Systems Transport, designed for multimedia and infotainment data

Attack surfaces_



V2V / V2G / V2L / V2X
Short / medium range communication

USB / ODB-II
Wired interfaces

Computer vision
ADAS

PLC
Communication

RFID / UHF
PKE / RKE

UHF
Direct TPMS

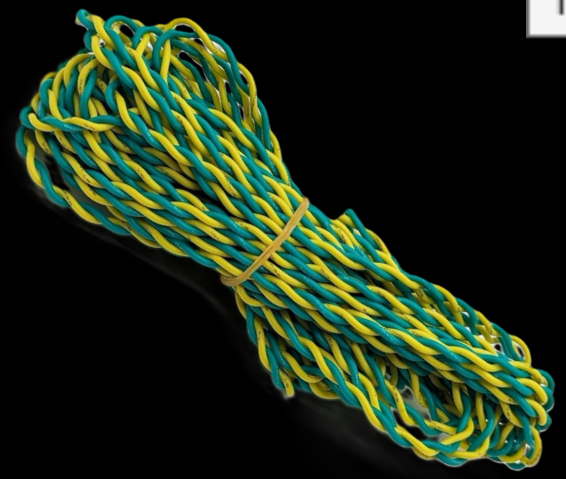
Automotive security_

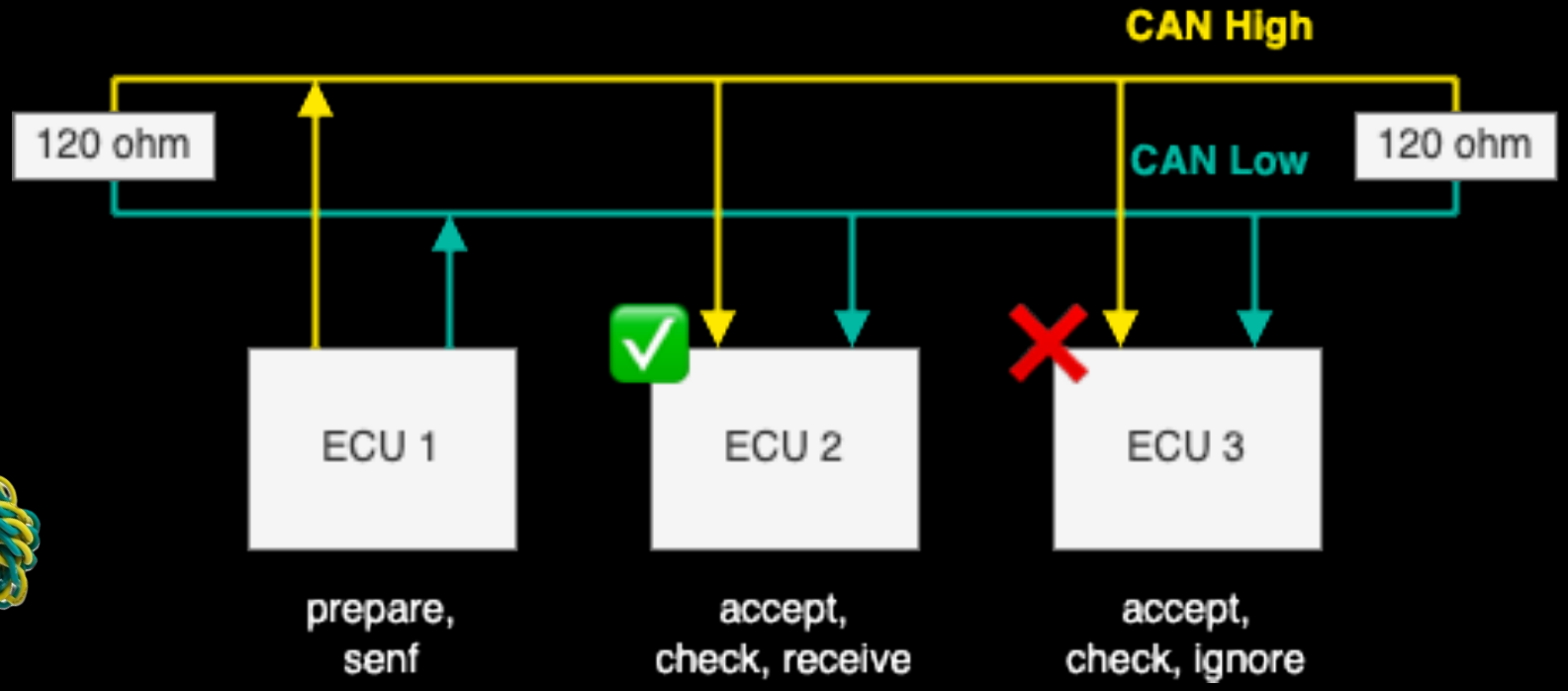
In **2015** security researchers shown **major vulnerabilities** in a Jeep, causing the recall of **1.5M** of cars in the US.

Hackers Remotely Kill a Jeep on a Highway by WIRED
https://www.youtube.com/watch?v=MK0SrxBC1xs

The CAN bus_

Controller Area Network (CAN) is a robust, real-time communication protocol designed for vehicles and industrial automation.





The CAN frame_

Contains 8 fields including:

- **ID:** the arbitration identifier, lower values have higher priority
- **Data:** data bytes aka payload
- …

CAN bus security flaws_

🔁 **No native protection against replay attacks**
Messages can be captured and resent without detection, allowing an attacker to repeat valid commands.

👤 **Lack of sender authentication**
The arbitration ID system does not verify the origin of a message, making it easy for an attacker to spoof a legitimate ECU.

💣 **Susceptibility to Denial-of-Service (DoS) attacks**
By injecting error frames deliberately, an attacker can force ECUs into bus-off state, effectively disabling them.

🔓 **No built-in encryption**
All data transmitted on the CAN bus is in plain text. Sensitive information (e.g., diagnostic data, text messages) can be easily intercepted and manipulated.

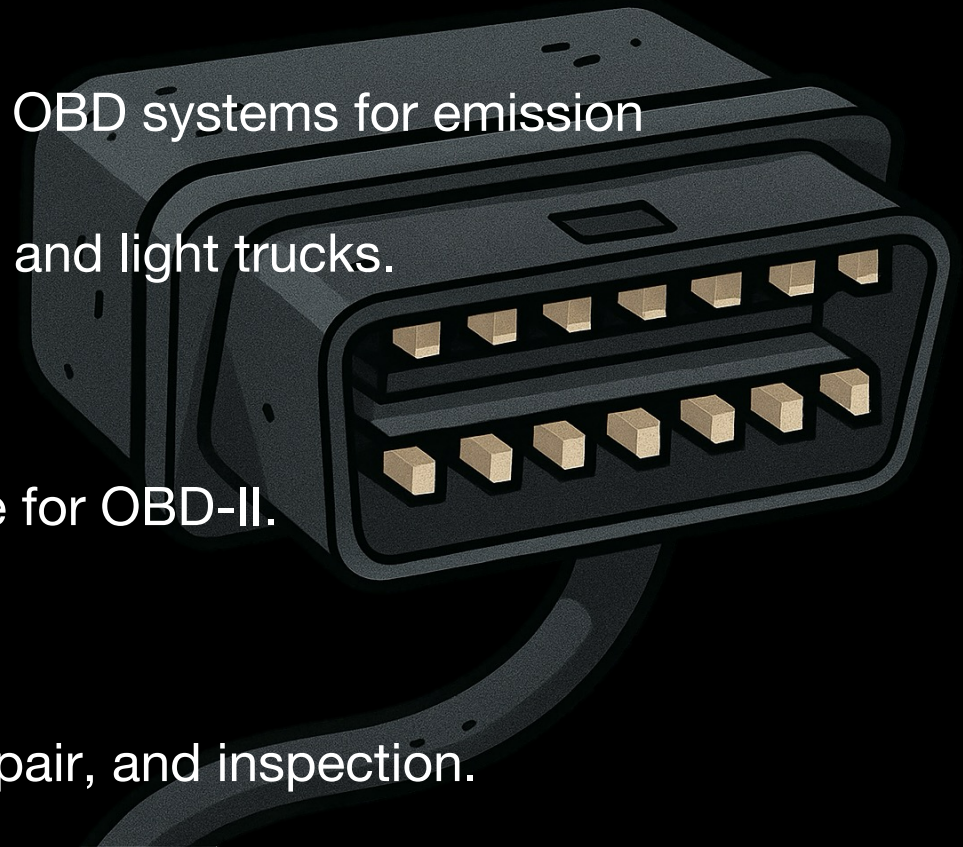
📣 **Broadcast nature of the bus**
Every ECU receives all messages, increasing the attack surface and making eavesdropping trivial.

ODB-II_

🔌 **Connector**
Standard 16-pin connector, typically located near the steering wheel.

🕰️ **History & milestones**

- **1991:** CARB (California Air Resources Board) required OBD systems for emission control.
- **1996**: OBD-II became mandatory in the US for all cars and light trucks.
- **2001:** Mandatory in the EU for gasoline vehicles.
- **2003**: Mandatory in the EU for diesel vehicles.
- **2008**: US vehicles must use CAN as the protocol base for OBD-II.



💡 **Why important?**
Enables emission monitoring and diagnostics.
Provides standardized access to car data for service, repair, and inspection.

Automotive security_

In April 2022 an automotive security researcher Ian Tabor tweeted that vandals had been at his car, pulling apart the headlight and unplugging cables.

CAN Injection: keyless car theft by Canis Automotive Labs
https://kentindell.github.io/2023/04/03/can-injection/

Discovering the CAN bus_

EXERCISE

Setting up a virtual CAN bus_

On Linux, you can create a virtual CAN bus using the ip command.

The commands to create and activate such a bus are shown below:

💻 Command

```
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
```

📌 Tip

Buses are commonly named based on their type:

- **vcan#** for virtual CAN interfaces (e.g., vcan0)
- **can#** for standard physical CAN buses (e.g., can0)
- **slcan#** for devices using the serial line CAN protocol (e.g., USB-to-CAN adapters running in SLCAN mode)

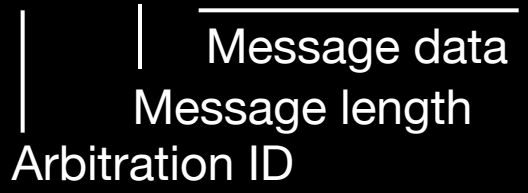
The can-utils library_

The **can-utils** package provides a collection of powerful tools for working with the CAN bus.

For example, the **candump** command can be used to capture and display all CAN messages on the vcan0 interface:

Command

```
$ candump vcan0
 vcan0  123   [4]  DE AD BE EF
```



Use the **cansend** command to transmit a single, custom CAN message:

Command

```
$ cansend vcan0 123#DEADBEEF
```

Use the **cangen** command to generate and send random CAN messages for testing purposes:

Command

```
$ cangen vcan0
```

Setting up a WebUSB CAN in Chrome_

Don’t want to install Linux and **can-utils**? Use the browser instead.

A browser-based CAN bus interface using the **WebUSB API**. No drivers, no installs — just open Chrome and connect your adapter.

Try it

roaldnefs.github.io/webusb-can/

Source

github.com/roaldnefs/webusb-can

📌 Requirements

- **Chrome 61+ or Edge 79+** — WebUSB is not supported in Firefox or Safari
- **HTTPS or localhost** — WebUSB requires a secure context
- **gs_usb / candleLight firmware** — e.g. CANable, CANable 2.0

WebUSBCAN

Simulator

Connection

Stop Simulator

Reset CAN

Bitrate: 500 kbit/s

Mode: Normal

- [x] Auto-scroll log
- [ ] Debug to console (F12)

Device Info

Vendor	Product
Simulator	Virtual CAN bus

VID:PID	Channels
—	1

Signals

Turn Left
0x188 | 01 00 00 00 00 00 00 00 | 50ms

Turn Right
0x188 | 02 00 00 00 00 00 00 00 | 50ms

Speed 100

RX 13565 179/s TX 0 0/s Errors 0

Log View | Pause | Clear Log | Export CSV

All | RX | TX | UDS

Filter by ID (hex, e.g. 7DF

ID	CYCLE (MS)	COUNT	DLC	DATA	ASCII
0AA	51.4	1469	[8]	00 00 00 00 19 00 00 00	....
174	1050.4	72	[8]	04 F7 07 7E 80 66 79 23	...~.fy#
188	51.5	1469	[8]	00 00 00 00 00 00 00 00	....
190	501.2	140	[8]	20 DD 66 3E 25 A5 5F F0	.f>%._.
19B	51.5	1469	[8]	00 00 00 00 00 00 00 00	....
206	200.7	325	[8]	A0 9E A8 D9 45 FC AE 06	....E...
244	51.4	1469	[8]	00 00 00 00 00 00 00 00	....
281	1050.5	72	[8]	0D 22 4F DD 53 F4 2D 15	."O.S.-.
2EF	1050.5	72	[8]	26 BF 4C 76 CD 6F 99 29	&.Lv.o.)
305	1050.4	72	[8]	81 D2 56 17 FB 65 96 8C	..V..e..
306	500.9	140	[8]	0F F2 75 68 93 EA 22 DD	..uh.".
324	200.5	325	[8]	3A 47 FA FB 7B 33 B3 05	:G..{3..
362	200.7	325	[8]	23 42 D0 A6 D4 2A CE 97	#B...*..
38D	101.2	578	[8]	F1 21 0B D4 9F 04 48 BC	.!....H.
3BA	200.6	325	[8]	22 07 B2 82 8B 16 C3 36	".....6
42A	200.7	325	[8]	3E EE 2E AE 89 55 3E 9D	>....U>.
432	101.3	578	[8]	8C 73 63 BA 59 23 77 54	.sc.Y#wT
450	101.3	578	[8]	21 CC 6B 54 FC 9E D2 2A	!.kT...*
479	500.9	140	[8]	6F 8F ED 96 89 49 42 14	o....IB.
488	101.0	578	[8]	F3 E1 E9 FB D6 2F 77 28	...../w(
4BC	51.0	954	[8]	7E 08 24 FD 35 1E BD B2	~.$.5...

VIRTUAL CLUSTER

◀ ▶

FL FR RL RR

Accel | Brake

Left signal | Right signal

FL | FR

RL | RR

- [x] Bogus traffic

7DF | 8 | Data bytes (hex, e.g. 02 01 0C 00 00 00 00 00) | Send | 100 ms | Repeat

The virtual car_

We'll use the **Instrument Cluster Simulator (ICSim)** by zombieCraig to simulate a car on our virtual CAN bus.

To download and compile ICSim, run the following commands:

Command

```
$ sudo apt install libsdl2-dev libsdl2-image-dev meson can-utils
$ git clone https://github.com/zombieCraig/ICSim.git
$ cd ICSim
$ meson setup builddir && cd builddir
$ meson compile
```

The builddir folder should now contain two executables: **controls** and **icsim**.

The virtual car_

⚠️ Troubleshooting: lib.o not linking

If you encounter the error:

/usr/bin/ld: lib.o: error adding symbols: file in wrong format

This is due to an architecture mismatch (for example, using an x86 object file on an ARM system). To fix it, follow these steps:

```
$ rm –r builddir
$ rm lib.o
$ gcc -c lib.c -o lib.o
$ meson setup builddir && cd builddir
```

The virtual car_

The **icsim** simulates the instrument cluster of our virtual car.

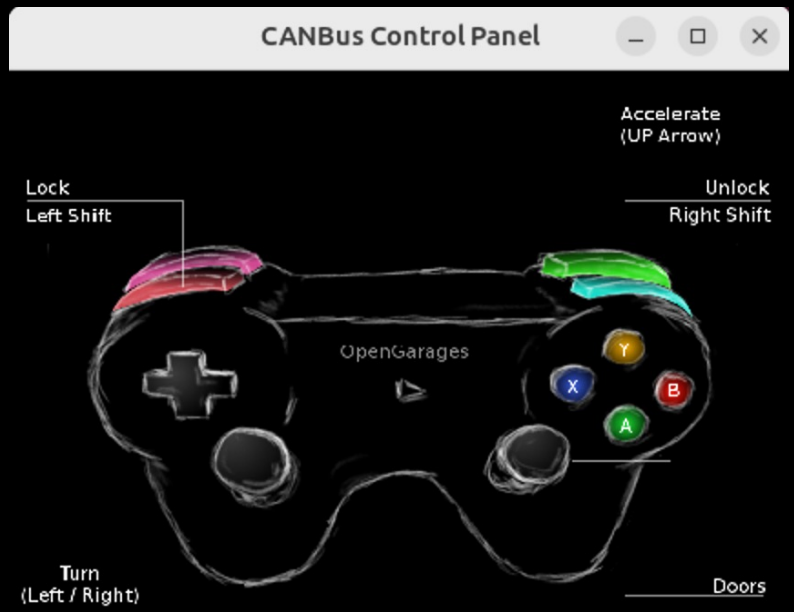
Command

```
$ ./icsim vcan0
```

The **controls** is used to operate (drive) our virtual car.

Command

```
$ ./controls vcan0
```



Find specific CAN message_

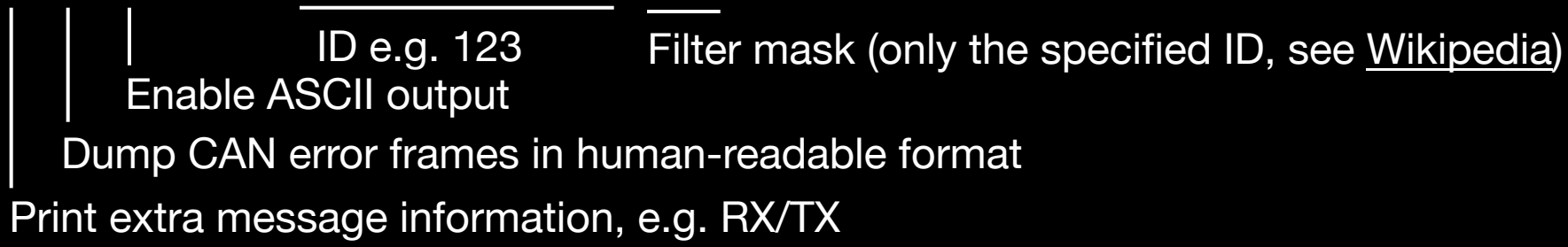
🎯 Goals

- Use the **controls** executable to activate the turn signals. Then, run **candump** to monitor messages on the **vcan0** bus. Take a few minutes to identify the **arbitration ID** associated with the turn signal messages.

  Answer:

📌 Tip

Use the following **candump** command to filter on a single arbitration ID:
$ candump -x -e -a vcan0,<Arbitration ID>:7FF



Find specific CAN message_

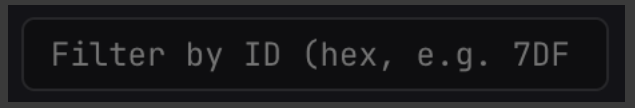
🎯 Goals

- Use the **simulator** to activate the turn signals. Then, open the **log view** to monitor messages on the CAN bus. Take a few minutes to identify the **arbitration ID** associated with the turn signal messages.

  Answer:

📌 Tip

Use the following filter input field to only show a single arbitration ID:



Sniffing CAN messages_

You might have noticed that it's quite difficult to find specific CAN messages using **candump** or **log view** alone. Fortunately, other tools like **cansniffer** or the **sniffer view** exist to make this process much easier and more efficient.

Command

```
$ cansniffer vcan0 -c
```

`I`
Color any changing bytes in the message

In **cansniffer** and **sniffer view**, any recurring messages that remain unchanged are automatically hidden after a few seconds. When a message does change, its data is updated on the same line, making it much easier to track active signals.

Tip

Make sure your terminal window is large enough to display all message lines clearly.

Sniffing CAN messages_

🎯 Goals

- Use the **controls** executable to activate the turn signals. Then, run **cansniffer** to monitor messages on the **vcan0** bus. Take a few minutes to identify the **arbitration ID** associated with the turn signal messages.

  Answer:

- Now, identify the message value used to activate the left turn signal.

  Answer:

- Then, repeat the process to find the value for the right turn signal.

  Answer:

📌 Tip

Use the **filtering** options in **cansniffer** to hide or display specific messages as needed.

Sniffing CAN messages_

🎯 Goals

- Use the **simulator** to activate the turn signals. Then, open the **sniffer view** to monitor messages on the CAN bus. Take a few minutes to identify the **arbitration ID** associated with the turn signal messages.

  Answer:

- Now, identify the message value used to activate the left turn signal.

  Answer:

- Then, repeat the process to find the value for the right turn signal.

  Answer:

📌 Tip

Use the **filtering** options in **sniffer view** to hide or display specific messages as needed.

Send CAN messages_

🎯 Goals

- Now, use **cansend** to transmit a turn signal message to the virtual instrument cluster. Write down the exact command you used.

  Answer:

- Use **cansend** to activate the hazard lights (both turn signals flashing together) on the virtual instrument cluster.

  Answer:

- Write a small shell script to make the hazard lights blink on and off repeatedly, simulating a real blinking sequence.

  Answer:

Send CAN messages_

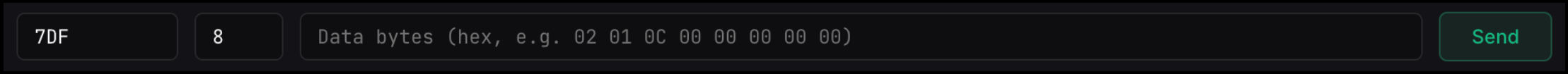
🎯 Goals

- Now, use the send panel at the bottom to transmit a turn signal message to the virtual instrument cluster. Write down the exact command you used.

  Answer:

- Use the send panel to activate the hazard lights (both turn signals flashing together) on the virtual instrument cluster.

  Answer:



Send periodic CAN messages_

You can use **cangen** to generate and send periodic CAN messages automatically.

This is useful for testing how an ECU or instrument cluster handles continuous traffic, stress-testing the bus, or simulating real vehicle activity.

cangen allows you to customize options such as the message ID range, data length, and interval (gap) between messages.

Command

```
$ cangen vcan0 -I 123 –L 8 –D DEADBEEF -g 100
```

-g: interval in ms

-D: the data to send

-L: data length

-I (upper i) to specify the arbitration ID

Send periodic CAN messages_

You can use **signals pannel** to generate and send periodic CAN messages automatically.

This is useful for testing how an ECU or instrument cluster handles continuous traffic, stress-testing the bus, or simulating real vehicle activity.

The **signals panel** allows you to customize options such as the message ID range, data length, and interval (gap) between messages.

Send CAN messages_

🎯 Goals

- Now, use **cangen** to make the hazard lights blink on and off repeatedly, simulating a real blinking sequence. Write down the exact command you used.

  Answer:

📌 Tip

This is particularly helpful for sending wake-up frames to keep the CAN bus and its connected ECUs active, preventing them from entering sleep mode.

Send CAN messages_

🎯 Goals

- Now, use **signals panel** to make the hazard lights blink on and off repeatedly, simulating a real blinking sequence. Write down the arbitration ID and payload.

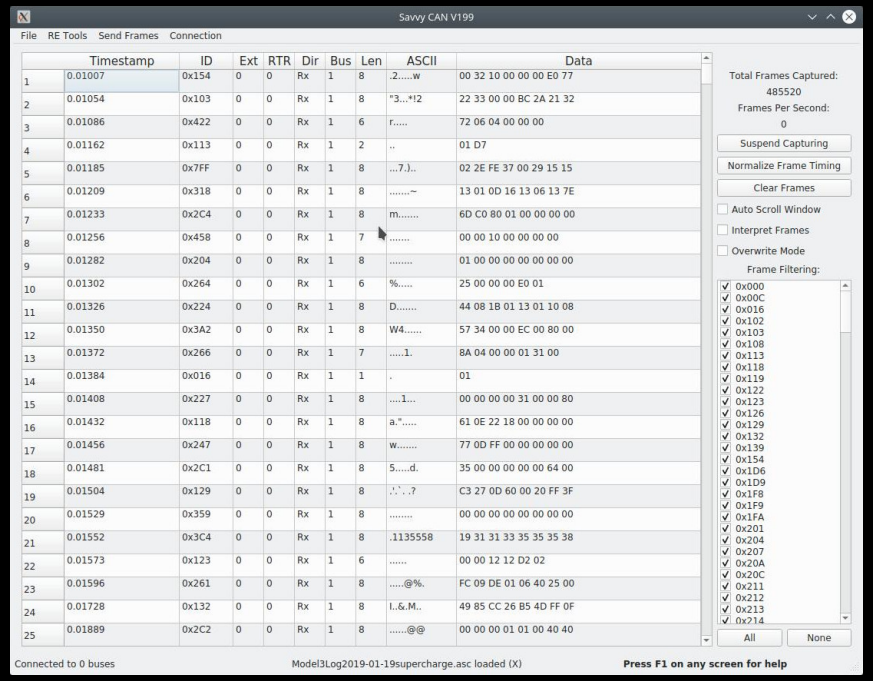
  Answer:

📌 Tip

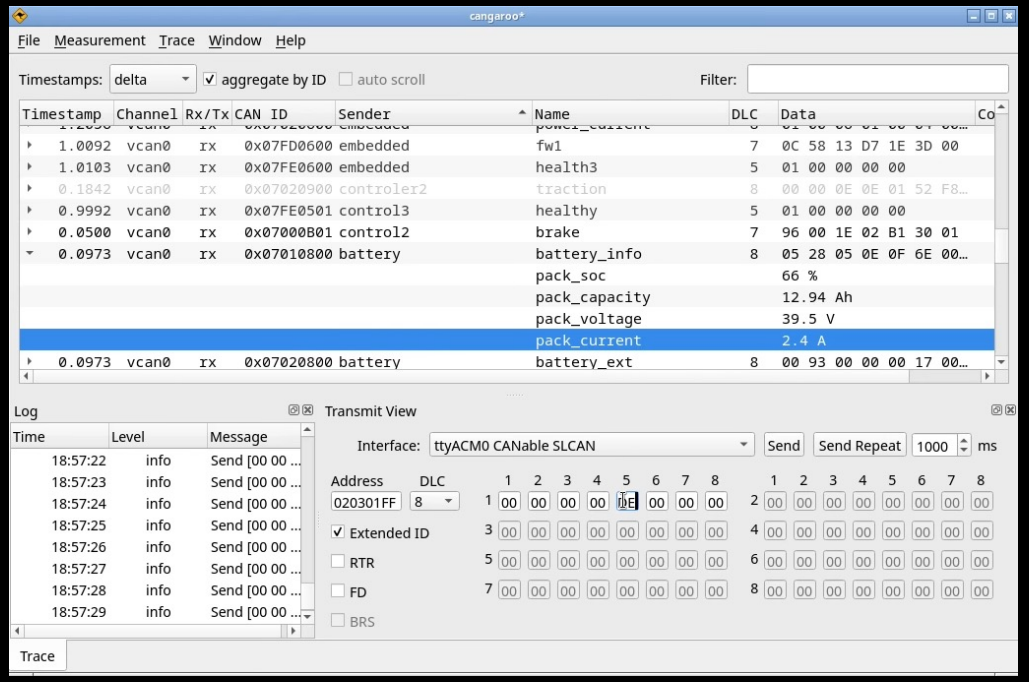
This is particularly helpful for sending wake-up frames to keep the CAN bus and its connected ECUs active, preventing them from entering sleep mode.

Using a GUI_

While we often use command-line tools like candump, cansend, and cansniffer, there are also several graphical tools that make it easier to monitor, analyze, and interact with CAN networks.



SavvyCan (Linux, Windows, and OSX)



Cangaroo (Linux, and Windows)

Fuzzing_

You can use **cangen** as a simple ‘poor man's fuzzer’.

Instead of specifying exact data values, you can use the **-i** (incremental) or **-r** (random) options to automatically generate changing payloads.

💻 Command

```
$ cangen vcan0 -I 123 –L 8 –D i -g 100
```

🎯 Goals

- Use **cangen** to send random signals to arbitration ID **0x19B**. Observe the virtual instrument cluster and describe what happens. What changes do you notice, and why do you think they occur?

  Answer:

Fuzzing_

You can use **fuzzing panel** as a simple ‘poor man's fuzzer’.

Instead of specifying exact data values, you can use the incremental or random options to automatically generate changing payloads.

🎯 Goals

- Use **fuzzing panel** to send random signals to arbitration ID **0x19B**. Observe the virtual instrument cluster and describe what happens. What changes do you notice, and why do you think they occur?

  Answer:

Fuzzing_

> ⚠️ **Warning**
>
> Be extremely careful when fuzzing a real vehicle!
>
> - Triggering critical safety functions unexpectedly (e.g., airbags, ABS).
> - Putting the vehicle into a fault or "limp" mode.
> - Bricking ECUs, especially if they receive a so-called "crash frame" or unexpected diagnostic command, which can require dealer-level tools to recover.
>
> Always perform fuzzing only in a **controlled lab environment**, on a test bench setup, or on a fully isolated vehicle system that is safe to reset.

Record and replay signals (optional)_

Waiting for the other groups to finish? Try these extra challenges in the meantime.

🎯 Goals

- Use **candump** with the **-l** option to save the captured CAN data to a file. pecify the exact command you ran, and indicate where the log file is stored.

  Answer:

- Use **canplayer** to replay the recorded CAN signals in smaller parts. This can help you reverse-engineer and analyze real CAN logs more effectively. Specify the exact command you used.

  Answer:

Manipulate an instrument cluster_

EXERCISE

Manipulate an instrument cluster_

👥 **Small Groups:**
Split into small groups. There is a limited number of instrument clusters, so make sure to give everyone the opportunity to take the primary role in each of the tasks.

🛠️ **Scenario:**
Imagine you are an attacker who has gained access to a car's **CAN bus** (for example, through an exposed diagnostic port or vulnerable ECU). Your goal is to send crafted CAN messages to manipulate the instrument cluster behavior.

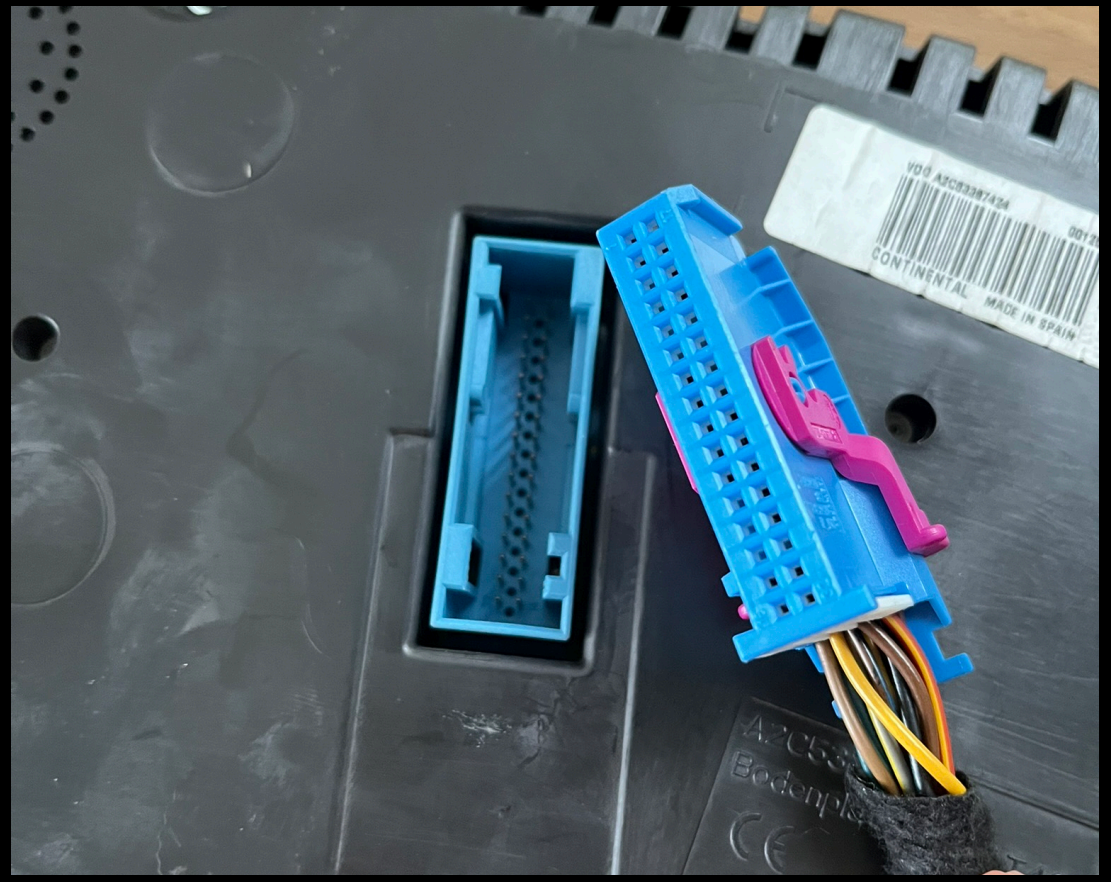
🎯 **Objectives:**
Understand how the instrument cluster receives and displays information from the CAN bus. Practice crafting and injecting CAN frames.

Instrument cluster_

For this workshop, we will be using instrument clusters from various Volkswagen Group cars produced around 2010. These include:

- Škoda Fabia
- SEAT Ibiza
- SEAT Leon

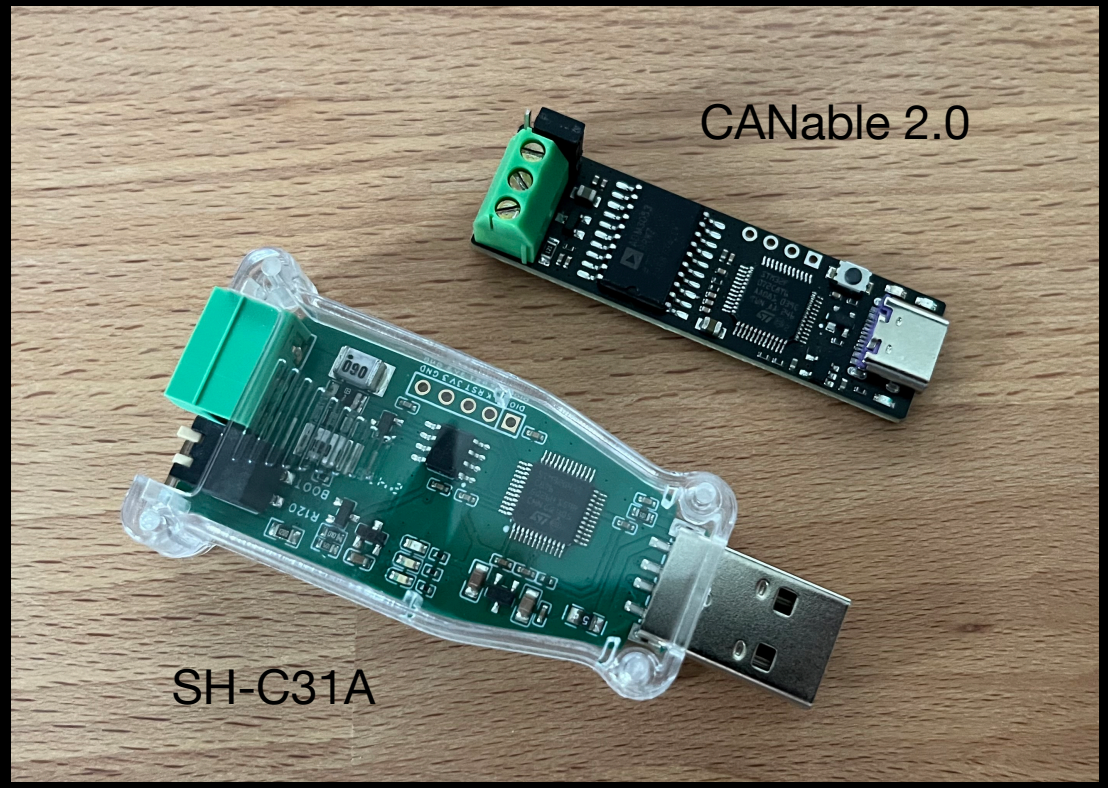
All of these clusters use a **single blue 32 pin connector**, making them easy to interface with for testing and security research.

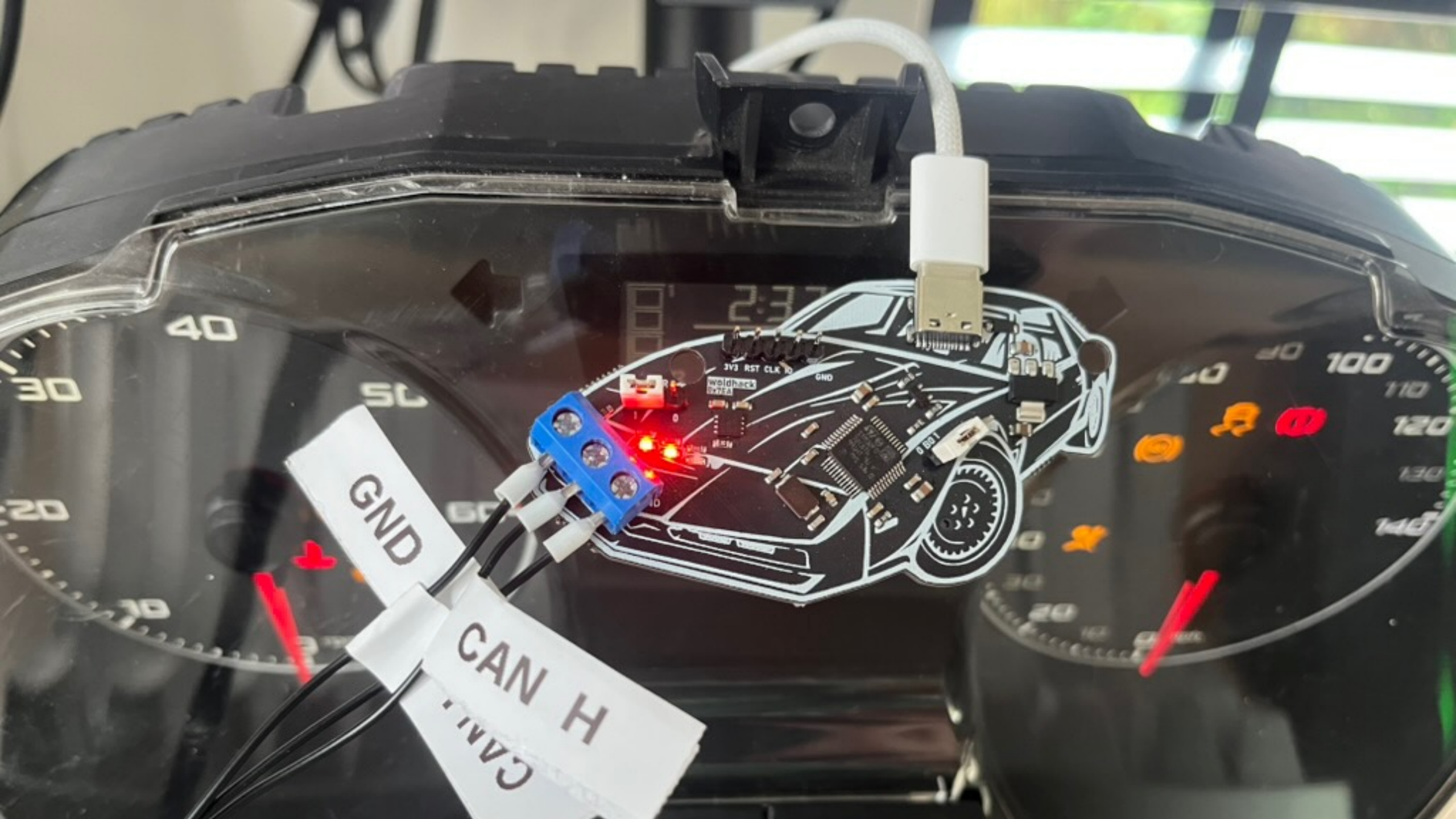


USB to CAN adapter_

During this workshop, we'll use the **CANable 2.0**, **SH-C31A** adapters or **BSides Groningen** badge to connect to the CAN bus.

- Compact and affordable USB-to-CAN adapters
- Open-source, widely used in automotive security research
- Support **CANtact**, **SocketCAN**, and **candlelight** firmware
- Compatible with Linux, Windows, and macOS
- Support high-speed CAN communication (up to 1 Mbps)




GND
CAN H
3V3 RST CLK IO
GND

Connector_

It’s relatively easy to build a **custom connector** or **harness** to interface with an instrument cluster for testing and security research.

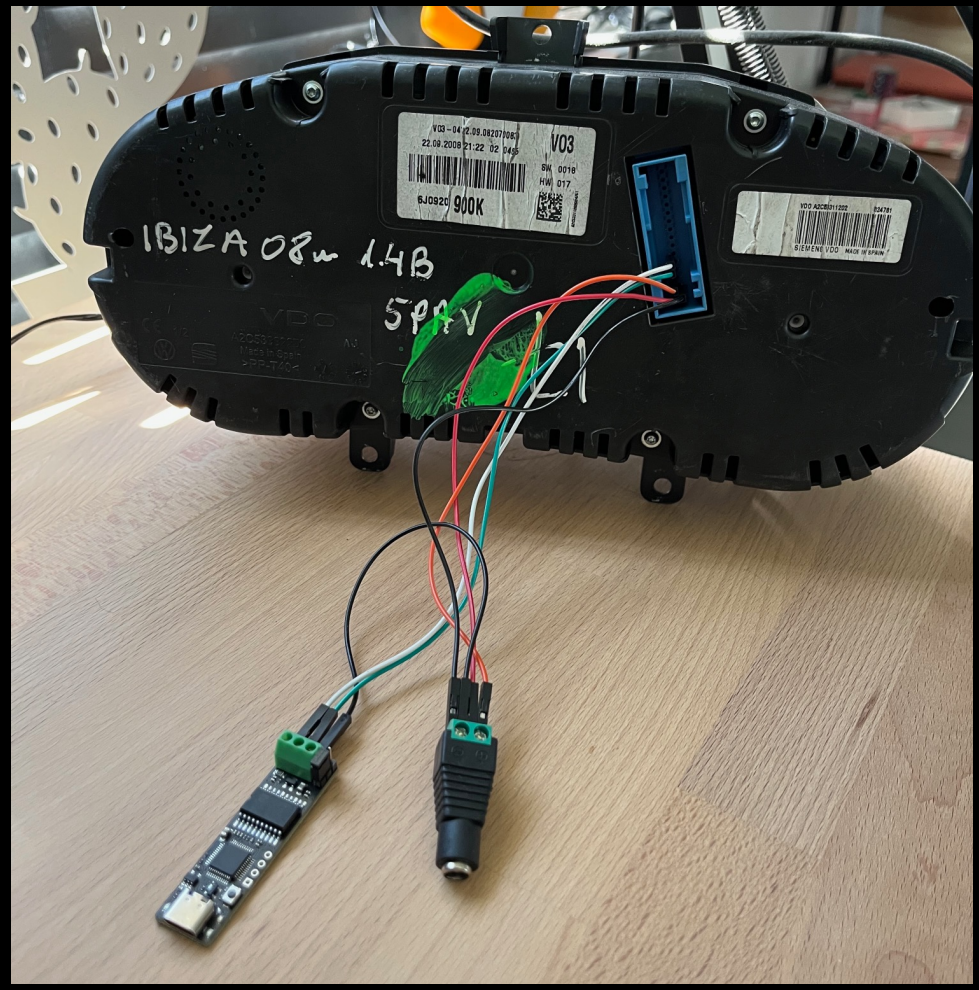
During this workshop, we will keep it simple and connect directly to the cluster using jump wires.

Connect to the cluster_

To connect to the instrument cluster, you'll need to:

1. Connect the USB-to-CAN adapter to the cluster's CAN bus.
2. Connect the USB-to-CAN adapter to your laptop.
3. Ensure power is correctly supplied to the cluster.

> ⚠ **Warning**
>
> Make sure to double-check all your connections before plugging in the computer and power supply. Incorrect wiring can damage the cluster.



Connect to the cluster_

📄 Pin layout

Pin 16: Ground
Pin 28: CAN high
Pin 29: CAN low
Pin 31: Switched ignition (12v)
Pin 32: Constant power (12v)

Source: https://mhhauto.com/Thread-need-pinout-cluster-MK6-golf

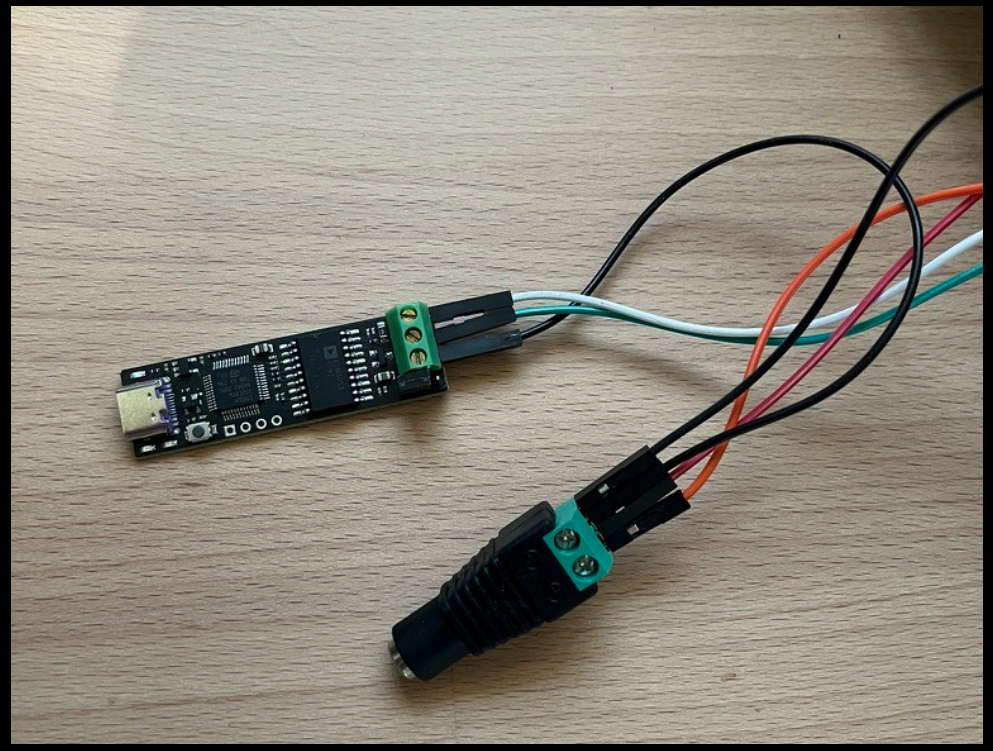
📌 Tip

The pin numbers are labeled directly on the connector. Using a light source can help you clearly see and identify the digits.

Connect to the cluster_

> ⚠️ **Warning**
>
> Make sure the **ground connection** of the USB to CAN adapter is connected to the ground connection of the power connector.

> 📌 **Tip**
>
> The power connector should have **two ground connections** and **two power connections**. This requires **multiple wires in a single socket**.



Connect to the cluster_

If the connection is correct the instrument cluster will show multiple warning lights and starts beeping.

On the USB-to-CAN adapter you should see multiple LEDs blinking to indicate the instrument cluster is sending data.

Question

Any ideas why the cluster might display multiple warnings and start beeping?

Connect to the cluster_

Configure the interface to interact with the USB-to-CAN adapter on can0:

💻 Command

```
$ ip link set can0 up type can bitrate 500000
```

🎯 Goals

- Now, use **candump** to listen for CAN messages from the instrument cluster. Make sure to record the exact command line you entered.

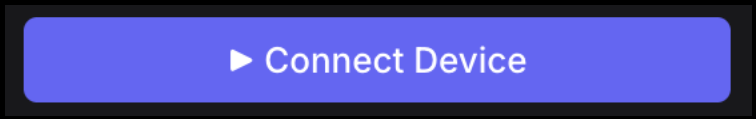
  Answer:

- Use **cansniffer** with color highlighting enabled to capture and analyze CAN messages from the instrument cluster. Record the exact command line you entered.

  Answer:

Connect to the cluster_

Click on the **Connect Device** button to connect the USB-to-CAN adapter:



🎯 Goals

- Now, use the **log view** to listen for CAN messages from the instrument cluster. Make sure to record one of the arbitration IDs send by the cluster.

  Answer:

- Use the **sniffer view** with to capture and analyze CAN messages from the instrument cluster. Record the exact command line you entered.

  Answer:

Spoof immobilizer_

An **immobilizer** is an electronic security device in cars designed to prevent the engine from starting unless the correct, authorized key or key fob is present.

On some vehicles, it is theoretically possible to **bypass or fake the immobilizer signal** by sending a crafted CAN message with arbitration ID **0x3D0** to the instrument cluster.

💻 Command

```
$ cangen can0 -I 3D0 -L 8 -D 0000000000000000 -g 100
```

🎯 Goals

- Use the above command and adjust the timing interval. At what interval (in milliseconds) must you repeatedly send the immobilizer message to keep the warning light off?

  Answer:

Spoof immobilizer_

An **immobilizer** is an electronic security device in cars designed to prevent the engine from starting unless the correct, authorized key or key fob is present.

On some vehicles, it is theoretically possible to **bypass or fake the immobilizer signal** by sending a crafted CAN message with arbitration ID **0x3D0** to the instrument cluster.

🎯 Goals

- Use the above input and adjust the timing interval. At what interval (in milliseconds) must you repeatedly send the immobilizer message to keep the warning light off?

  Answer:

Research_

📌 **Tip**

There are some differences between instrument clusters. The SEAT Leon uses different arbitration IDs and data values to achieve the same signals as the SEAT Ibiza and Skoda Fabia.

🎯 **Goals**

- Research online to find information about CAN messages used in the **PQ24**, **PQ34**, and **PQ35** platforms. Which platform does your instrument cluster belong to?

  Answer:

- What are the **arbitration IDs** of the CAN messages used to control the turn signals and display the RPM speed?

  Answer:

Send turn and RPM signals_

Hopefully, you've found some information about the correct arbitration IDs to use. In practice, you would typically analyze a **CAN log from a real car** to reverse-engineer these IDs and their corresponding data values. If that information isn't available, you can always fall back on **fuzzing the instrument cluster** to discover the correct IDs and signals.

🎯 Goals

- Identify the correct arbitration ID and data value needed to activate the **turn signals**. Then, use **cangen** to send this command to the instrument cluster. Which exact command did you use?

  Answer:

- Repeat the process for the **RPM signal**. Can you make the instrument cluster display the redline? What exact command did you use?

  Answer:

Send turn and RPM signals_

Hopefully, you've found some information about the correct arbitration IDs to use. In practice, you would typically analyze a **CAN log from a real car** to reverse-engineer these IDs and their corresponding data values. If that information isn't available, you can always fall back on **fuzzing the instrument cluster** to discover the correct IDs and signals.

🎯 Goals

- Identify the correct arbitration ID and data value needed to activate the **turn signals**. Then, use **signals panel** to send this signal to the instrument cluster. Which exact arbitration ID and payload did you use?

  Answer:

- Repeat the process for the **RPM signal**. Can you make the instrument cluster display the redline? What arbitration ID and payload did you use?

  Answer:

Connect ICSim to cluster (optional)_

Let’s connect the ICSim controller to the actual instrument cluster.

🎯 Goals

- Connect the **ICSim controller** to the same CAN bus as the instrument cluster. Are you able to successfully send turn signal commands? If not, why do you think they might not work?

  Answer:

- Begin by carefully analyzing (reverse-engineering) the **ICSim controller source code**. Then, update or adjust the required sections to ensure the code functions as intended.

  Answer:

Send other signals (optional)_

Waiting for the other groups to finish? Try these extra challenges in the meantime.

🎯 Goals

- Can you turn off the other warning lights? Which exact arbitration IDs and data values do you need to send to achieve this? Write them down.

  Answer:

- Are you also able to send the **car speed signal**? You should be able to find additional information about the required arbitration IDs and data values online.

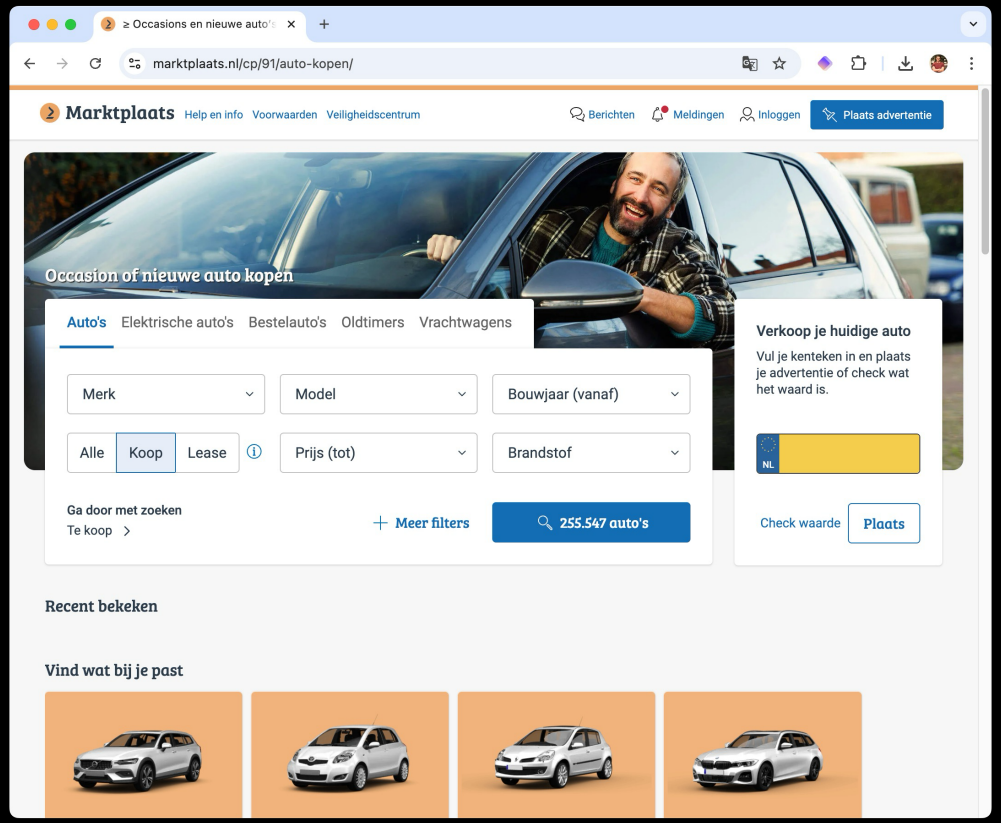
  Answer:

- (🐦) Implement your solution using the python-can library to both send messages to the instrument cluster and receive feedback from it.

  Answer:

Regroup_

Car in a box_


Occasions en nieuwe auto's
marktplaats.nl/cp/91/auto-kopen/
Marktplaats
Help en info
Voorwaarden
Veiligheidscentrum
Berichten
Meldingen
Inloggen
Plaats advertentie
Occasion of nieuwe auto kopen
Auto's
Elektrische auto's
Bestelauto's
Oldtimers
Vrachtwagens
Merk
Model
Bouwjaar (vanaf)
Alle
Koop
Lease
Prijs (tot)
Brandstof
Ga door met zoeken
Te koop
Meer filters
255.547 auto's
Verkoop je huidige auto
Vul je kenteken in en plaats je advertentie of check wat het waard is.
NL
Check waarde
Plaats
Recent bekeken
Vind wat bij je past


makita
x17
B-52059

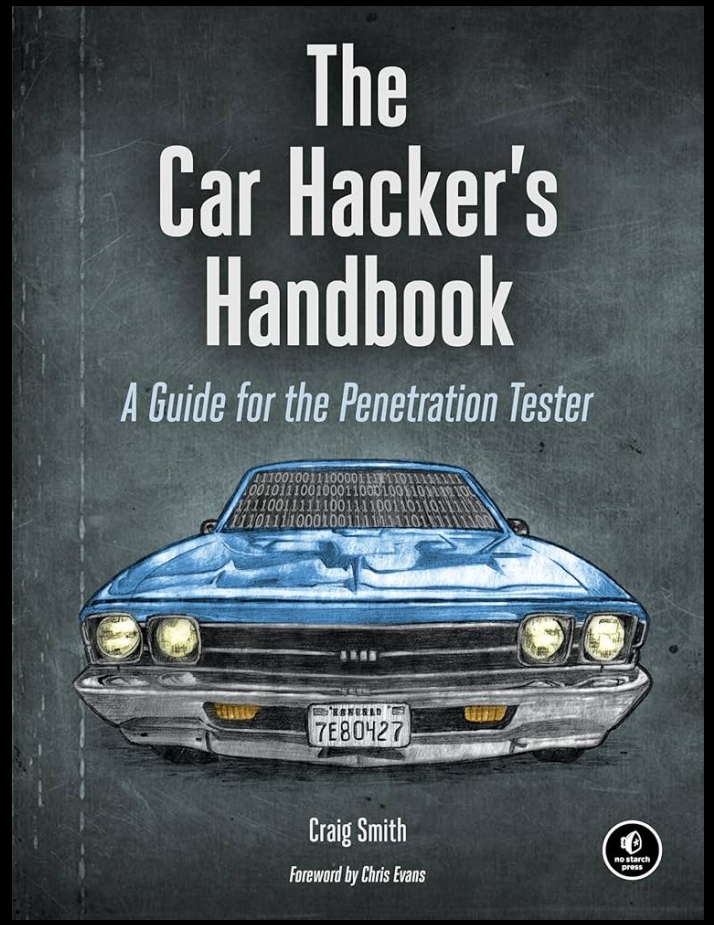
Tesla in a box_

Image source
https://nl.wikipedia.org/wiki/Tesla_Model_3

Resources_

Some additional resources:

- reddit.com/r/CarHacking
- Book: The Car Hacker’s Handbook
- Whitepaper: Car Hacking: For Poories
- Defcon: Car Hacking Village
- …



BSides Groningen & Amsterdam_

Thanks_

Hands-on Car Hacking & Automotive Cybersecurity
Workshop by Roald Nefs

Do you have any questions?
roald.nefs@warpnet.nl
linkedin.com/in/roaldnefs/
warpnet.nl