March 2023 - https://djnn.sh

Car Hacking 101

01

HOW TO GET STARTED IN AUTOMOTIVE SECURITY ?
A SOFTWARE ENGINEER'S PERSPECTIVE.

BY DJNN

whoami

DJNN --- HTTPS://DJNN.SH/ABOUT

- CS student from France interested in Cybersecurity
- Mostly interested back-end / systems stuff
- C, Golang, Python, Haskell, Elixir (struggling at learning Rust)
- Not a lot of experience with hardware

02
Presentation Overview

Attack surface overview

Overview of the different attack surfaces we can find in a car, along with a brief overview of the history of the automotive cybersecurity landscape.

How to get started ?

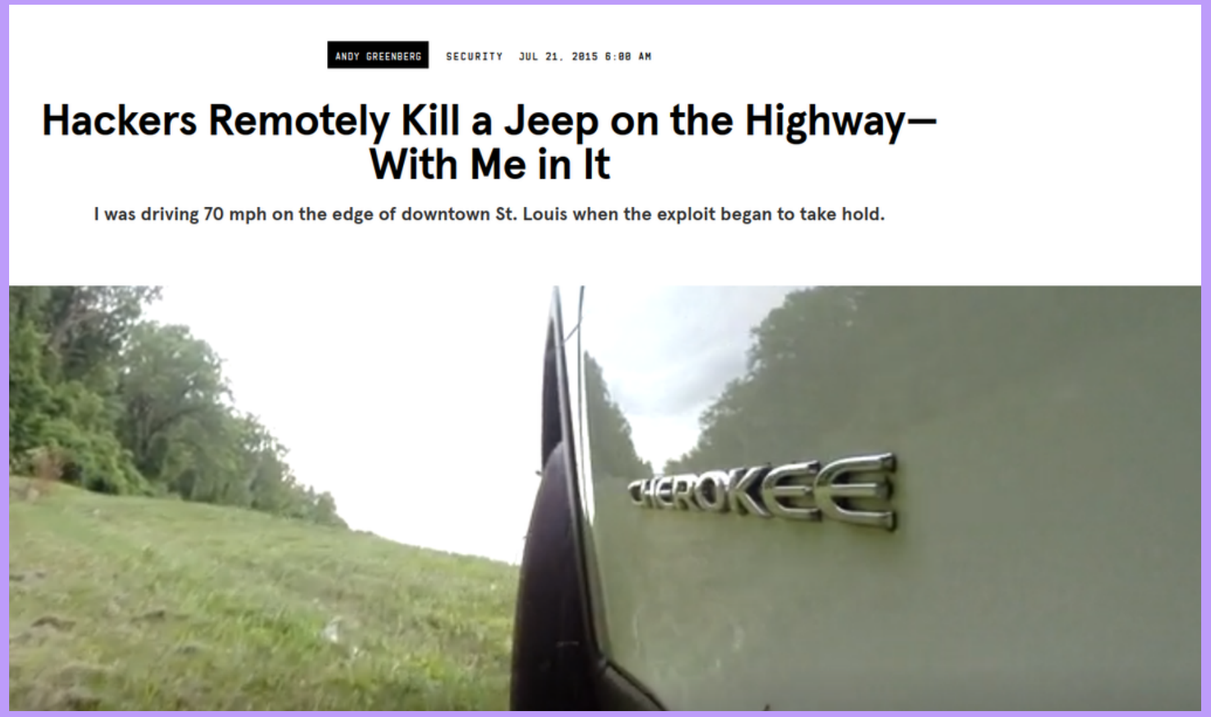
What kind of tooling do you need to get started ? Where to look first ? Description of the main protocols used in automotive systems.

Tips and tricks

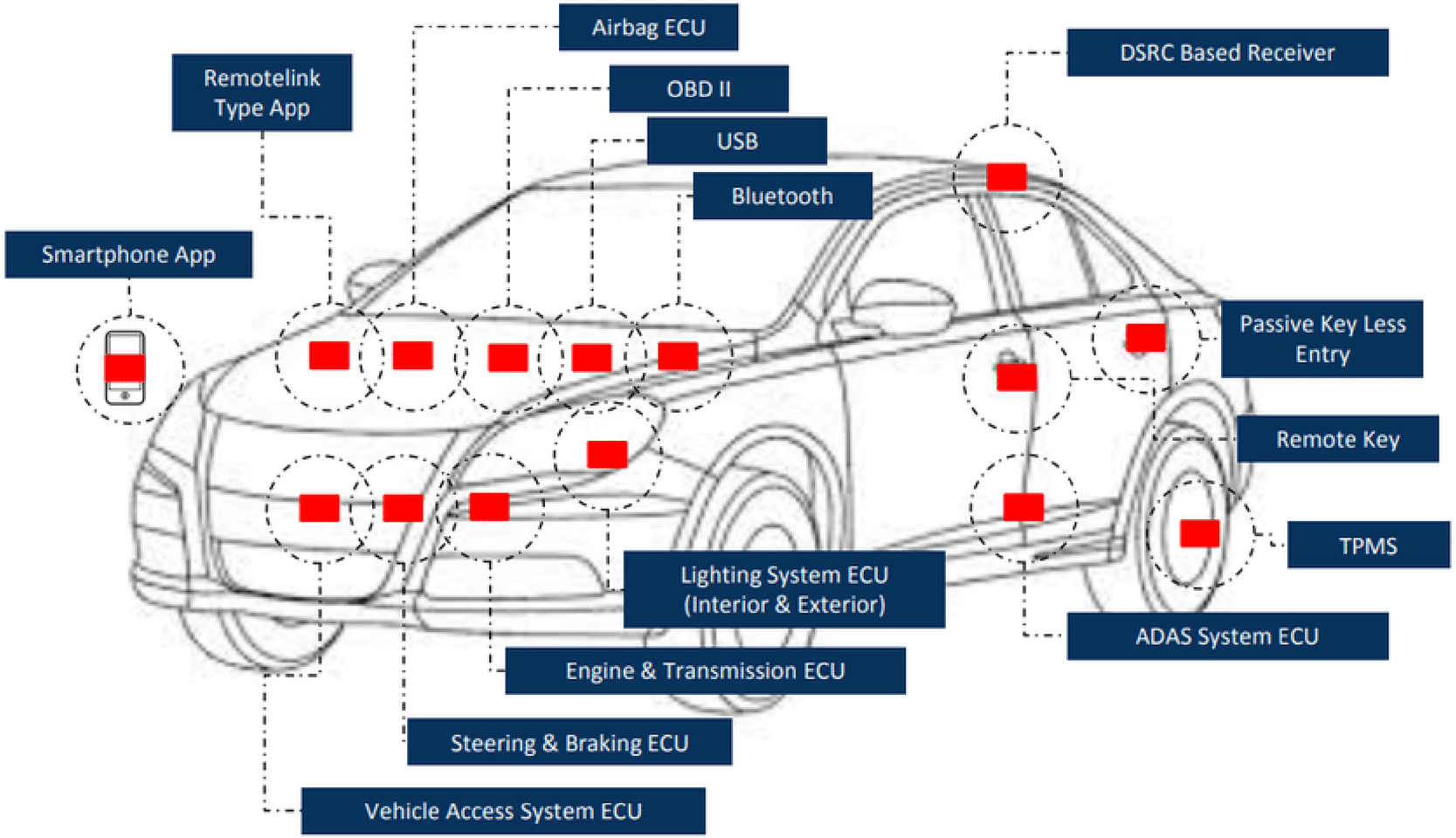
Where do you get information regarding Car systems ?
How can you perform attacks.
Overview of the last important CVEs you should be looking at.

Catching up...

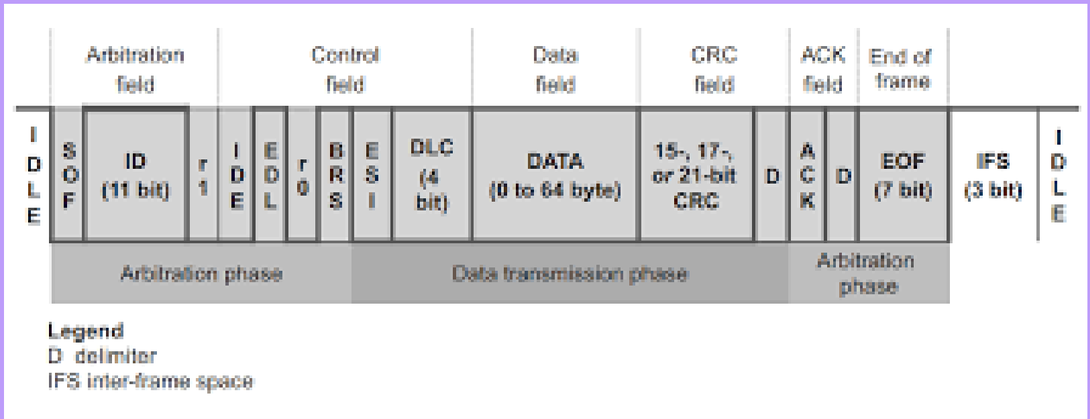
- car alarms & security systems have been around since 1920s
- mostly mechanical until end of 1990s
- automotive security for OEMs wasn't a thing until 2015



Attack surfaces



CAR PROTOCOLS (CAN / CAN-FD)



- Referred to as CAN bus
- Designed by Bosch about 50 years ago
- Very reliable and flexible
- No authentication required to listen on the bus
- Efforts done recently to encrypt messages using SecOc (Secure On-Board Communications)

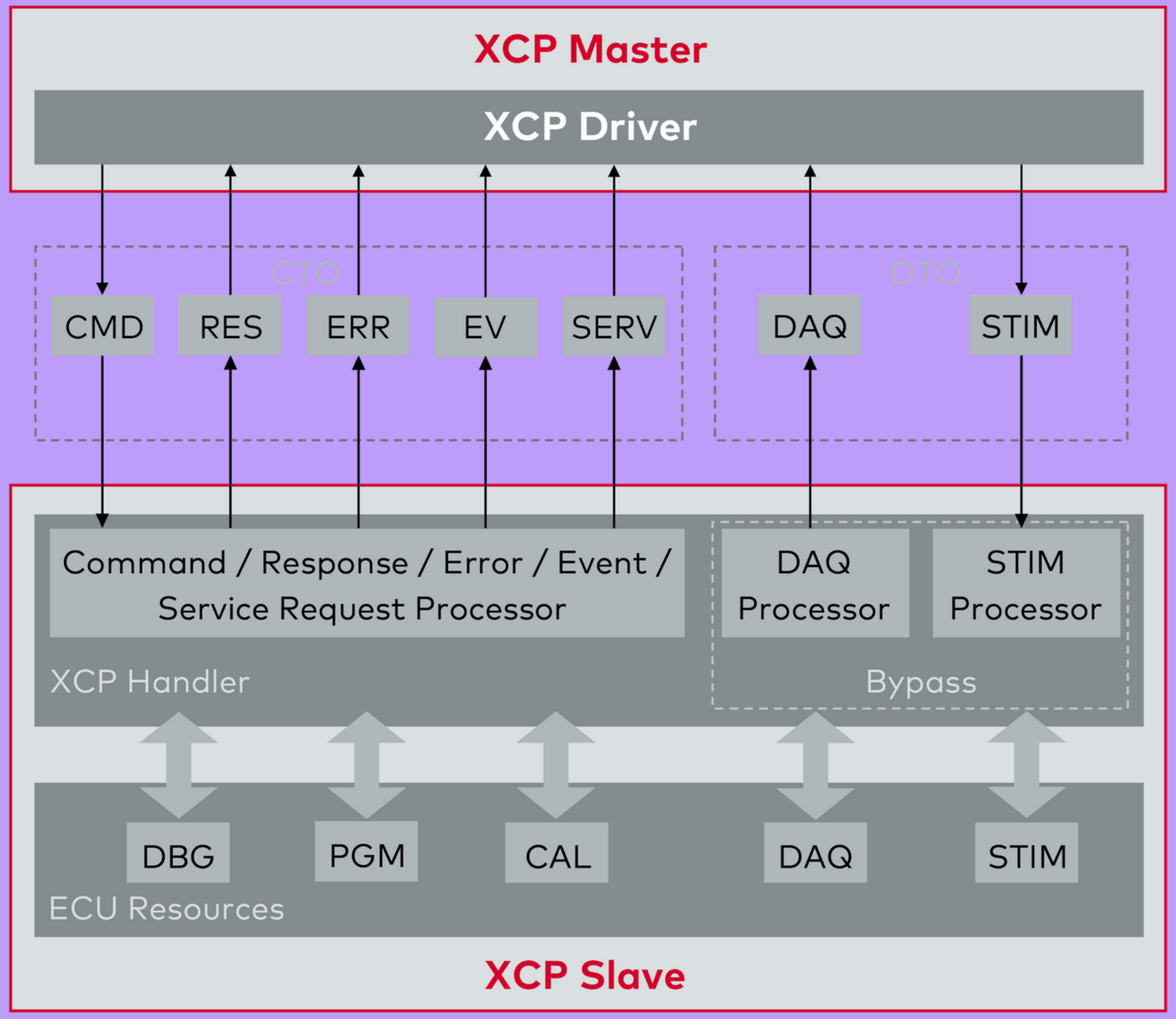
```
#!/bin/python3

import can

bus = can.Bus() : Bus
while True:
    msg = can.Message(12, data=[0 for _ in range(8)]) : Message
    bus.send(msg)
```

CAR PROTOCOLS (UDS)

UDS service identifiers (SIDs)

	UDS SID (request)	UDS SID (response)	Service	Details
Diagnostic and Communications Management	0x10	0x50	Diagnostic Session Control	Control which UDS services are available
	0x11	0x51	ECU Reset	Reset the ECU ("hard reset", "key off", "soft reset")
	0x27	0x67	Security Access	Enable use of security-critical services via authentication
	0x28	0x68	Communication Control	Turn sending/receiving of messages on/off in the ECU
	0x29	0x69	Authentication	Enable more advanced authentication vs. 0x27 (PKI based exchange)
	0x3E	0x7E	Tester Present	Send a "heartbeat" periodically to remain in the current session
	0x83	0xC3	Access Timing Parameters	View/modify timing parameters used in client/server communication
	0x84	0xC4	Secured Data Transmission	Send encrypted data via ISO 15764 (Extended Data Link Security)
	0x85	0xC5	Control DTC Settings	Enable/disable detection of errors (e.g. used during diagnostics)
	0x86	0xC6	Response On Event	Request that an ECU processes a service request if an event happens
	0x87	0xC7	Link Control	Set the baud rate for diagnostic access
Data Transmission	0x22	0x62	Read Data By Identifier	Read data from targeted ECU - e.g. VIN, sensor data values etc.
	0x23	0x63	Read Memory By Address	Read data from physical memory (e.g. to understand software behavior)
	0x24	0x64	Read Scaling Data By Identifier	Read information about how to scale data identifiers
	0x2A	0x6A	Read Data By Identifier Periodic	Request ECU to broadcast sensor data at slow/medium/fast/stop rate
	0x2C	0x6C	Dynamically Define Data Identifier	Define data parameter for use in 0x22 or 0x2A dynamically
	0x2E	0x6E	Write Data By Identifier	Program specific variables determined by data parameters
	0x3D	0x7D	Write Memory By Address	Write information to the ECU's memory
DTCs	0x14	0x54	Clear Diagnostic Information	Delete stored DTCs
	0x19	0x59	Read DTC Information	Read stored DTCs, as well as related information
	0x2F	0x6F	Input Output Control By Identifier	Gain control over ECU analog/digital inputs/outputs
	0x31	0x71	Routine Control	Initiate/stop routines (e.g. self-testing, erasing of flash memory)
Upload/ Download	0x34	0x74	Request Download	Start request to add software/data to ECU (incl. location/size)
	0x35	0x75	Request Upload	Start request to read software/data from ECU (incl. location/size)
	0x36	0x76	Transfer Data	Perform actual transfer of data following use of 0x74/0x75
	0x37	0x77	Request Transfer Exit	Stop the transfer of data
	0x38	0x78	Request File Transfer	Perform a file download/upload to/from the ECU
		0x7F	Negative Response	Sent with a Negative Response Code when a request cannot be handled

CAR PROTOCOLS (XCP)



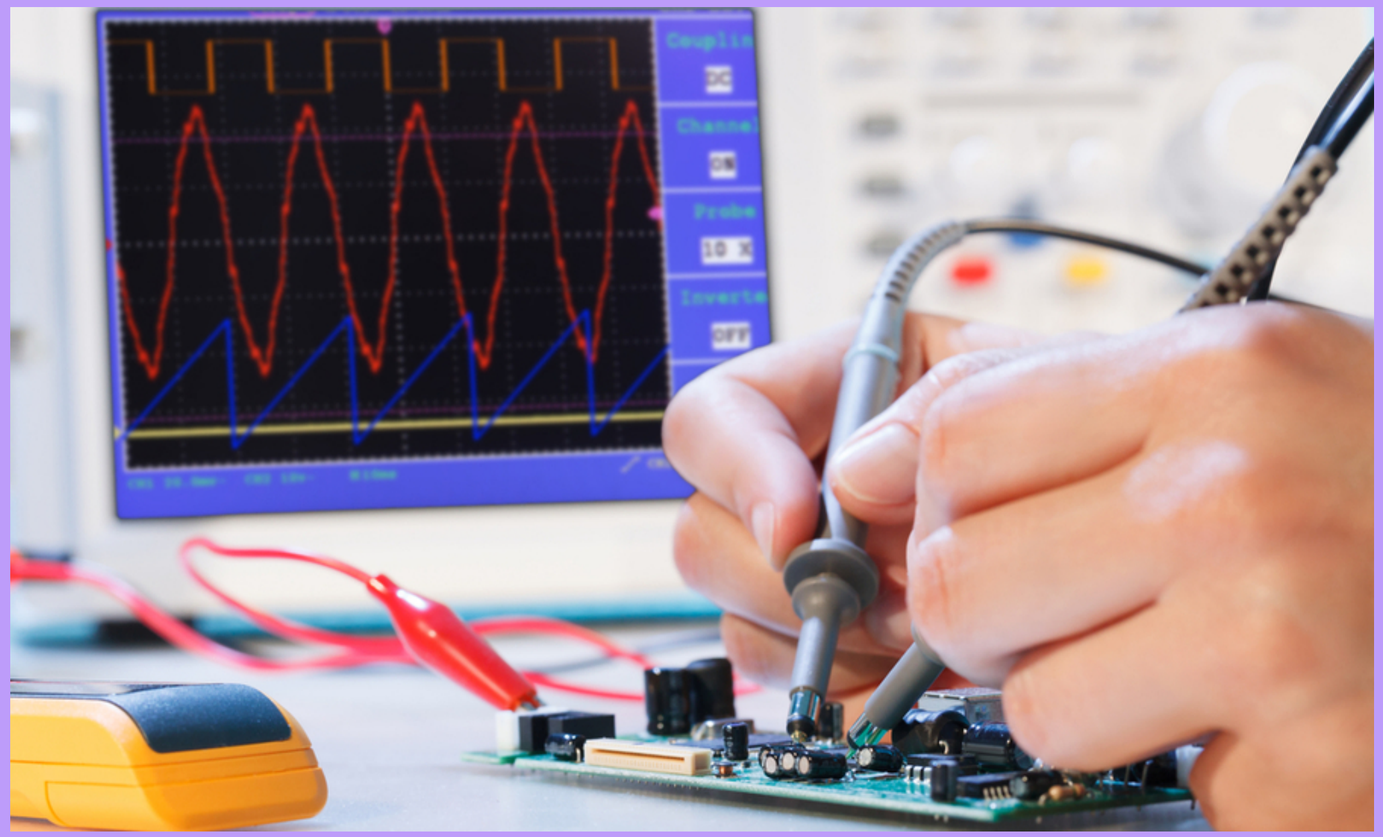
- Mostly useful for debugging and calibrations
- Usually password-protected
- allows for arbitrary memory-read

SUPPLY CHAIN

Firmware can be delivered
OTA (over the air)
or by aftermarket repair shops

-> OTA support depends a lot on OEMs

HARDWARE ATTACKS (TLDR)



GETTING STARTED

Enterprise-level

- can simulate a whole car
- supports all types of protocols (XCP, CAN/CAN-FD, Flexray, UDS)
- Fully integrated suite
- Ultra-mega expansive

Researcher tooling

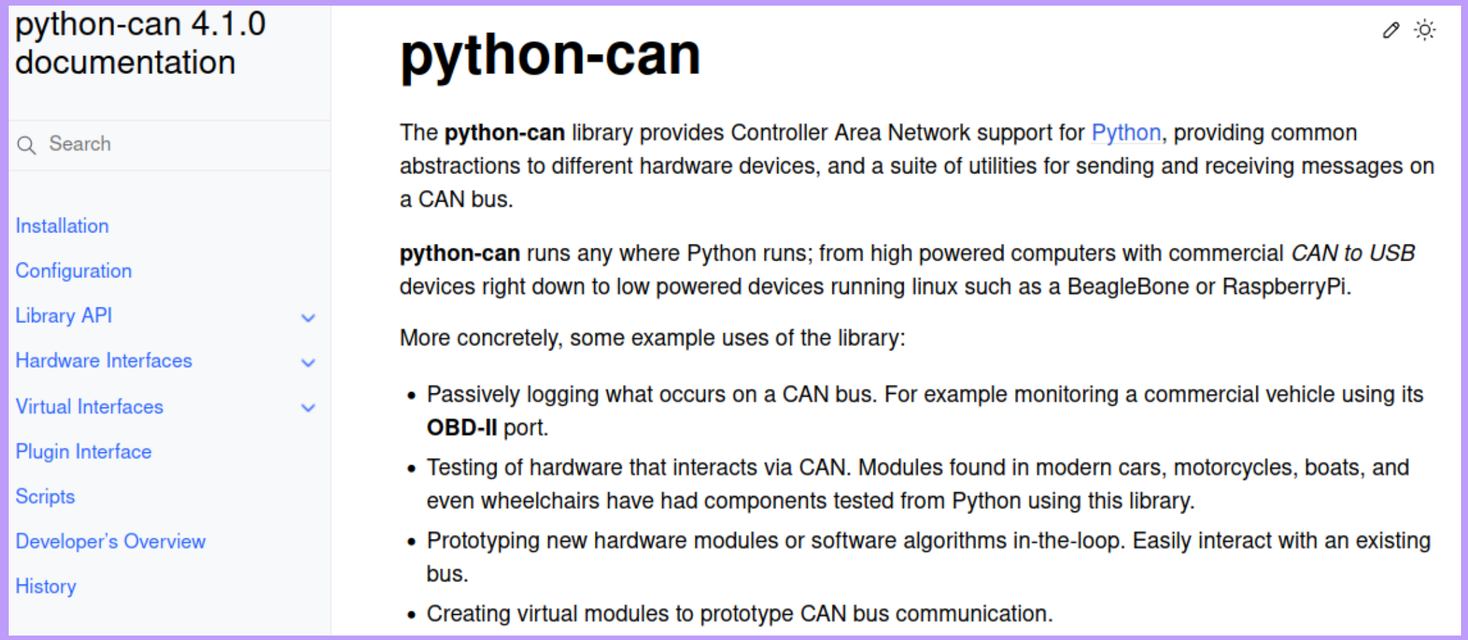
- can range from very cheap, to a bit expansive
- Usually only support CAN/CAN-FD
- Some can comunicate wirelessly
- Cannot do RF stuff

CAR HACKING AND LINUX

-> Supported since Linux 4.5
-> no dependencies required ! (though can-utils is a useful package)
-> Support for CAN-FD is available through additional configuration

All you need is Python and any type of connector to get started

(Metasploit actually has some Car hacking modules but I have not tried them)



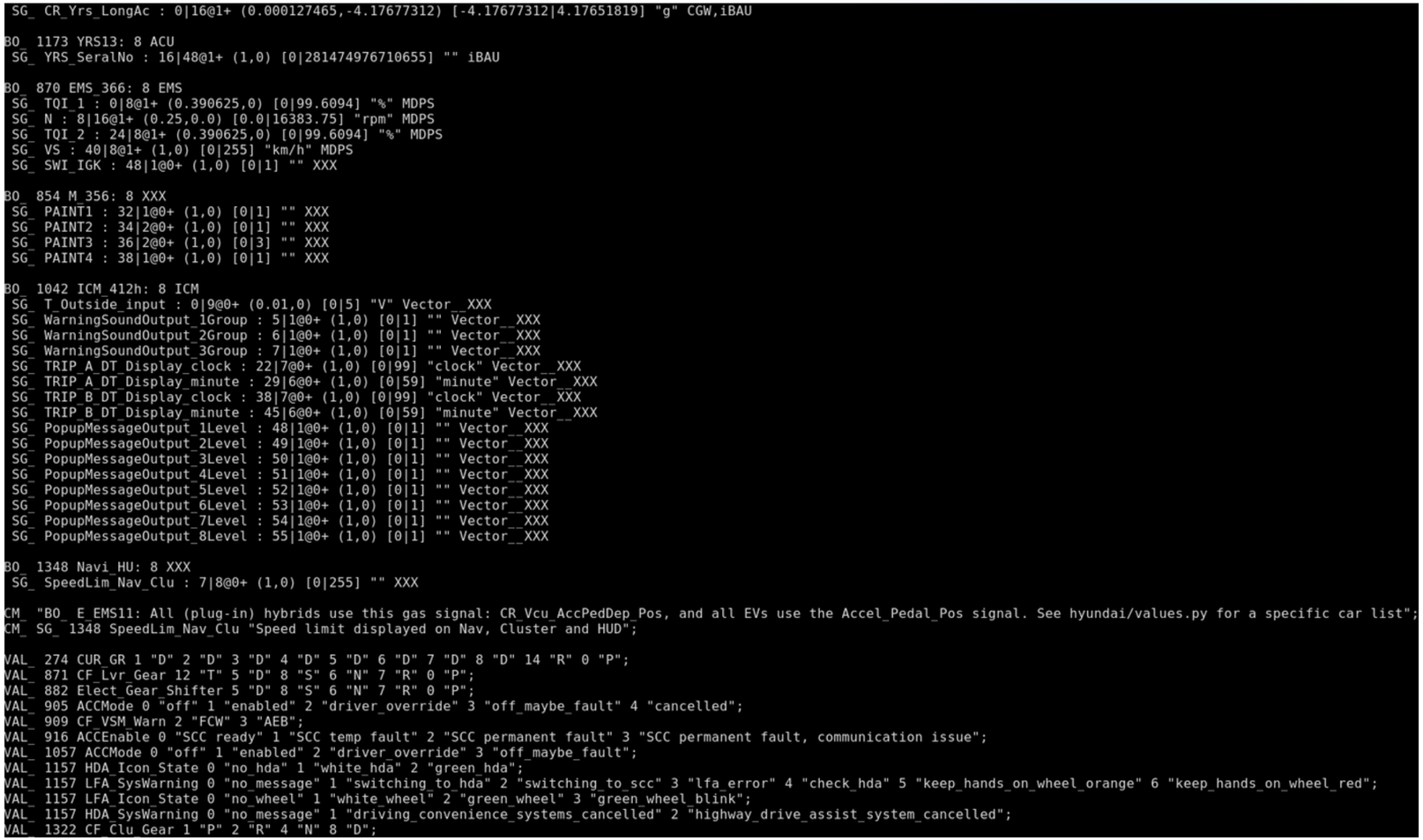

```
~ sudo apt install can-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
can-utils is already the newest version (2020.11.0-1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
~ sudo modprobe vcan
~ sudo ip link add dev vcan0 type vcan
~ sudo ip link set up vcan0
```

CAN databases

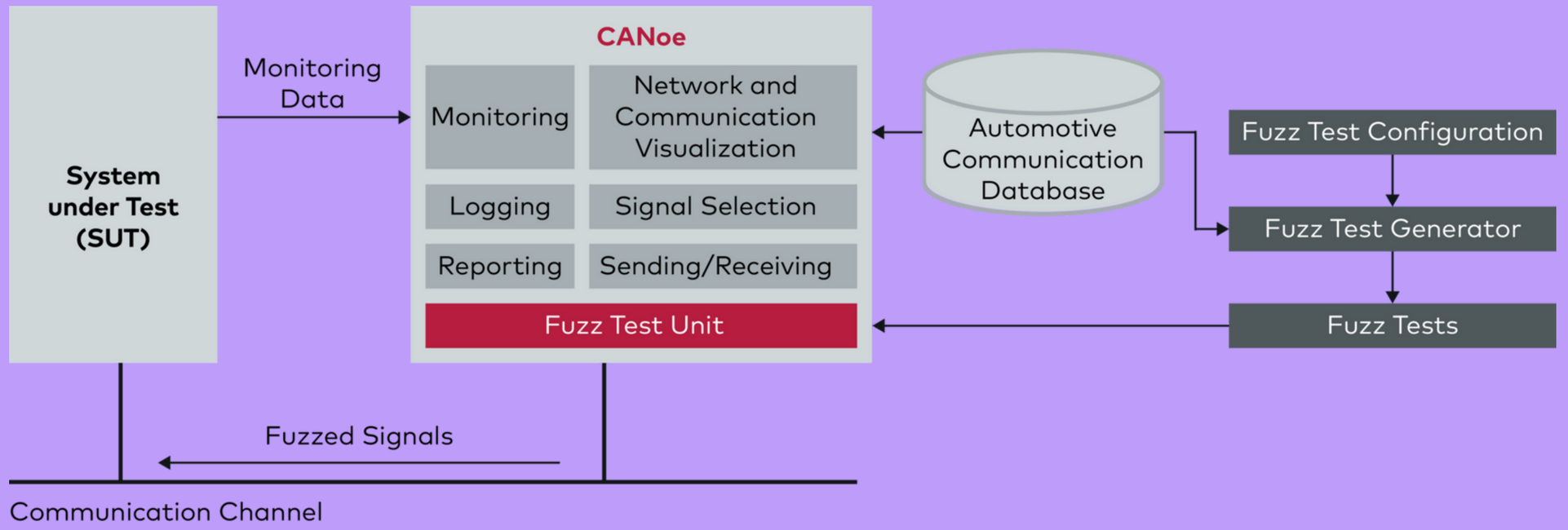
Each OEM uses different message and checksum algorithms, which are often stored in CAN *databases*

Most common formats are *.dbc, .REF*

Depending on models, you can find some that have already been reverse-engineered



```
 SG_ CR_Yrs_LongAc : 0|16@1+ (0.000127465,-4.17677312) [-4.17677312|4.17651819] "g" CGW,iBAU

BO_ 1173 YRS13: 8 ACU
 SG_ YRS_SeralNo : 16|48@1+ (1,0) [0|281474976710655] "" iBAU

BO_ 870 EMS_366: 8 EMS
 SG_ TQI_1 : 0|8@1+ (0.390625,0) [0|99.6094] "%" MDPS
 SG_ N : 8|16@1+ (0.25,0.0) [0.0|16383.75] "rpm" MDPS
 SG_ TQI_2 : 24|8@1+ (0.390625,0) [0|99.6094] "%" MDPS
 SG_ VS : 40|8@1+ (1,0) [0|255] "km/h" MDPS
 SG_ SWI_IGK : 48|1@0+ (1,0) [0|1] "" XXX

BO_ 854 M_356: 8 XXX
 SG_ PAINT1 : 32|1@0+ (1,0) [0|1] "" XXX
 SG_ PAINT2 : 34|2@0+ (1,0) [0|1] "" XXX
 SG_ PAINT3 : 36|2@0+ (1,0) [0|3] "" XXX
 SG_ PAINT4 : 38|1@0+ (1,0) [0|1] "" XXX

BO_ 1042 ICM_412h: 8 ICM
 SG_ T_Outside_input : 0|9@0+ (0.01,0) [0|5] "V" Vector__XXX
 SG_ WarningSoundOutput_1Group : 5|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ WarningSoundOutput_2Group : 6|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ WarningSoundOutput_3Group : 7|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ TRIP_A_DT_Display_clock : 22|7@0+ (1,0) [0|99] "clock" Vector__XXX
 SG_ TRIP_A_DT_Display_minute : 29|6@0+ (1,0) [0|59] "minute" Vector__XXX
 SG_ TRIP_B_DT_Display_clock : 38|7@0+ (1,0) [0|99] "clock" Vector__XXX
 SG_ TRIP_B_DT_Display_minute : 45|6@0+ (1,0) [0|59] "minute" Vector__XXX
 SG_ PopupMessageOutput_1Level : 48|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_2Level : 49|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_3Level : 50|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_4Level : 51|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_5Level : 52|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_6Level : 53|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_7Level : 54|1@0+ (1,0) [0|1] "" Vector__XXX
 SG_ PopupMessageOutput_8Level : 55|1@0+ (1,0) [0|1] "" Vector__XXX

BO_ 1348 Navi_HU: 8 XXX
 SG_ SpeedLim_Nav_Clu : 7|8@0+ (1,0) [0|255] "" XXX

CM_ "BO_ E_EMS11: All (plug-in) hybrids use this gas signal: CR_Vcu_AccPedDep_Pos, and all EVs use the Accel_Pedal_Pos signal. See hyundai/values.py for a specific car list";
CM_ SG_ 1348 SpeedLim_Nav_Clu "Speed limit displayed on Nav, Cluster and HUD";

VAL_ 274 CUR_GR 1 "D" 2 "D" 3 "D" 4 "D" 5 "D" 6 "D" 7 "D" 8 "D" 14 "R" 0 "P";
VAL_ 871 CF_Lvr_Gear 12 "T" 5 "D" 8 "S" 6 "N" 7 "R" 0 "P";
VAL_ 882 Elect_Gear_Shifter 5 "D" 8 "S" 6 "N" 7 "R" 0 "P";
VAL_ 905 ACCMode 0 "off" 1 "enabled" 2 "driver_override" 3 "off_maybe_fault" 4 "cancelled";
VAL_ 909 CF_VSM_Warn 2 "FCW" 3 "AEB";
VAL_ 916 ACCEnable 0 "SCC ready" 1 "SCC temp fault" 2 "SCC permanent fault" 3 "SCC permanent fault, communication issue";
VAL_ 1057 ACCMode 0 "off" 1 "enabled" 2 "driver_override" 3 "off_maybe_fault";
VAL_ 1157 HDA_Icon_State 0 "no_hda" 1 "white_hda" 2 "green_hda";
VAL_ 1157 LFA_SysWarning 0 "no_message" 1 "switching_to_hda" 2 "switching_to_scc" 3 "lfa_error" 4 "check_hda" 5 "keep_hands_on_wheel_orange" 6 "keep_hands_on_wheel_red";
VAL_ 1157 LFA_Icon_State 0 "no_wheel" 1 "white_wheel" 2 "green_wheel" 3 "green_wheel_blink";
VAL_ 1157 HDA_SysWarning 0 "no_message" 1 "driving_convenience_systems_cancelled" 2 "highway_drive_assist_system_cancelled";
VAL_ 1322 CF_Clu_Gear 1 "P" 2 "R" 4 "N" 8 "D";
```



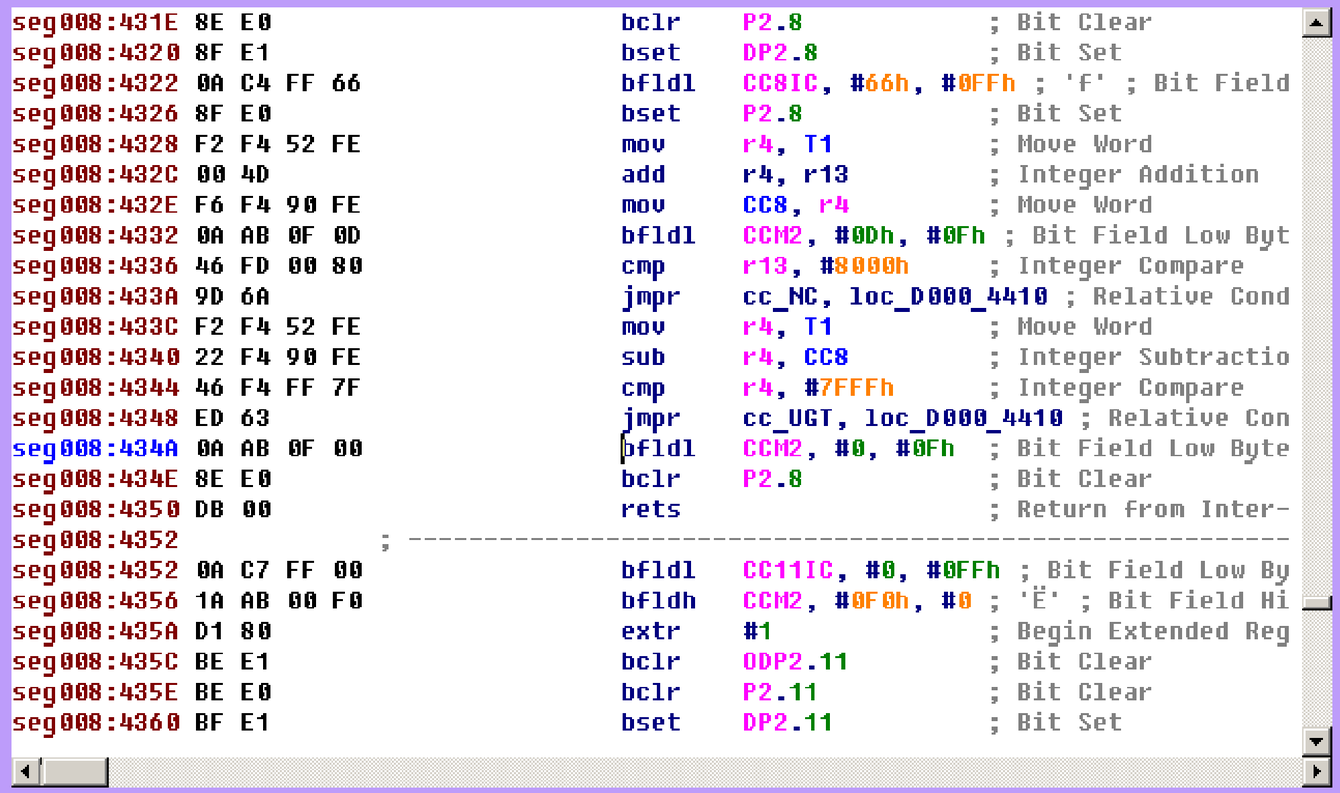

```
❯ python3 cc.py -i can3 uds_fuzz seed_randomness_fuzzer

------------------
CARING CARIBOU v0.3
------------------

Loaded module 'uds_fuzz'

usage: cc.py uds_fuzz seed_randomness_fuzzer [-h] [-t ITERATIONS] [-r RTYPE] [-id RTYPE]
                                             [-m RMETHOD] [-d D]
                                             stype src dst
cc.py uds_fuzz seed_randomness_fuzzer: error: the following arguments are required: stype, src, dst
```

Good write-ups

- Tesla Synacktiv Pwn2Own: https://www.synacktiv.com/sites/default/files/2022-10/tesla_hexacon.pdf
- Another Zero-click Tesla RCE: https://kunnamon.io/tbone/tbone-v1.0-redacted.pdf
- Tales of a former Tesla employee: https://forums.somethingawful.com/showthread.php?threadid=3862643&userid=0&perpage=40&pagenumber=62
- How I hacked my car: https://programmingwithstyle.com/posts/howihackedmycar/
- Defcon 28 Car Hacking Village: https://www.carhackingvillage.com/defcon28talks
- Honda / Nissan car hacking write-up: https://twitter.com/samwcyo/status/1597792097175674880

Reverse-engineering

From experience, IDA is the best decompiler for CAR firmware because it supports many architectures out of the box



Threat intellingence

Lot of RE efforts available online for big OEMs, you just have to look for them! :

Example: https://github.com/ludwig-v/psa-nac-firmware-reverse-engineering

Interesting projects

Libraries: Scapy, python-can, python cantools

Software projects: comma-ai, Wireshark, CaringCaribou, CANToolZ

Documentation: CAN-utils, OpenDBC, OpenVehicles

Thank you

GET IN TOUCH

Website

https://djnn.sh

Github

https://github.com/bogdzn

Email address

email@djnn.sh