CAR Hacking Workshop
CAN Bus Exploitation

Yogesh Ojha

GreHack
New is not always better

Agenda - Theory

- Introduction to Hardware and Software System in a vehicle
- Introduction to vehicle communication network - CAN
- Attack Surface Mapping on a Vehicle System

Agenda - Practical

- Hands-on with SocketCAN, can-utils and Wireshark
- Diving deep into ICSim
- Intro to SavvyCAN and Macchina M2

Goal

The goal of this workshop is to help you get started with Car hacking fast, easy and cheap.

This is to help more people clear the entry barrier in Car Hacking.

More importantly all this, you will be able to practice in the simulator on your favorite Linux Distribution without worrying to break your car.

/Users/yogeshojha/GreHack> whoami

USER INFORMATION

Yogesh Ojha
From Nepal
Cyber Security Analyst
Tata Consultancy Services India

Primary Research area includes
IoT Security, Hardware Hacking
and mobile application security
Speaker at HITB Abu Dhabi, NoConName Spain,Kazakhstan Kazakhstan, Open Source Summit China, FOSS Asia Summit Singapore, etc
---

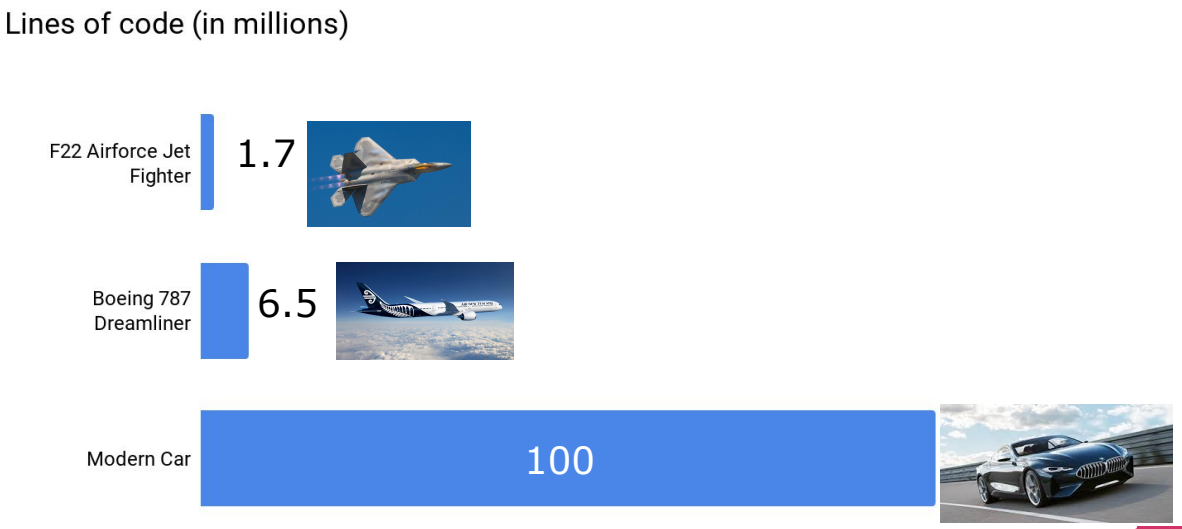
Medium
https://medium.com/@yogeshojha

TATA CONSULTANCY SERVICES

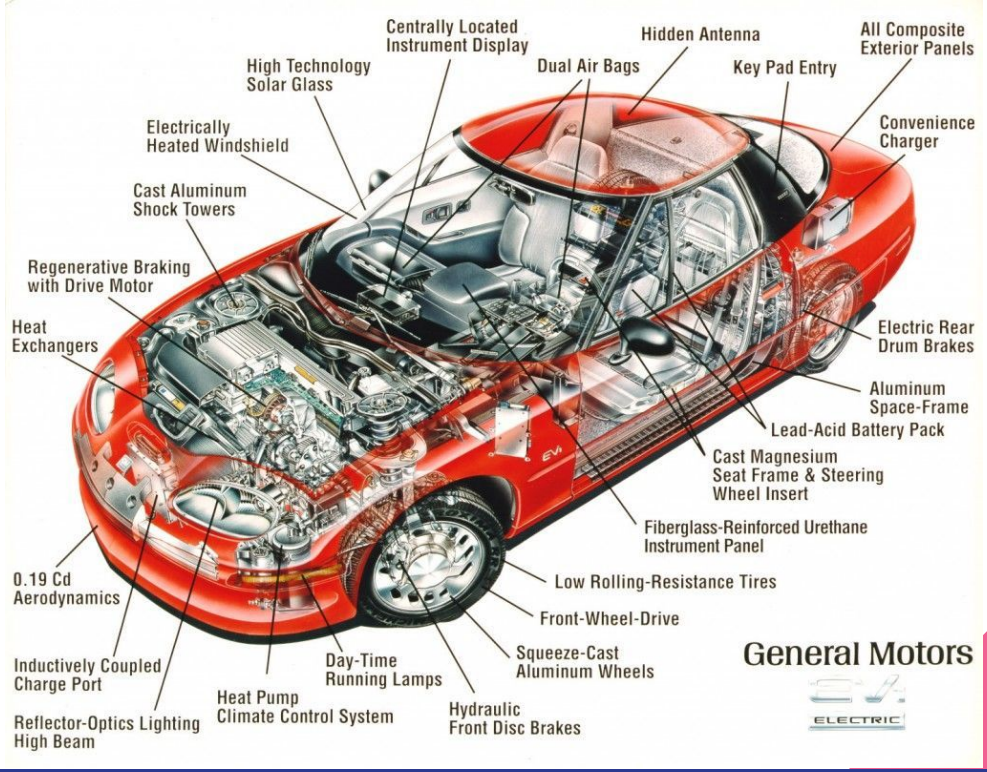
How does a modern car function?

When you are driving a car today, you are driving a hugely powerful computer that happens to have wheels and steering.

Complexity in a modern car

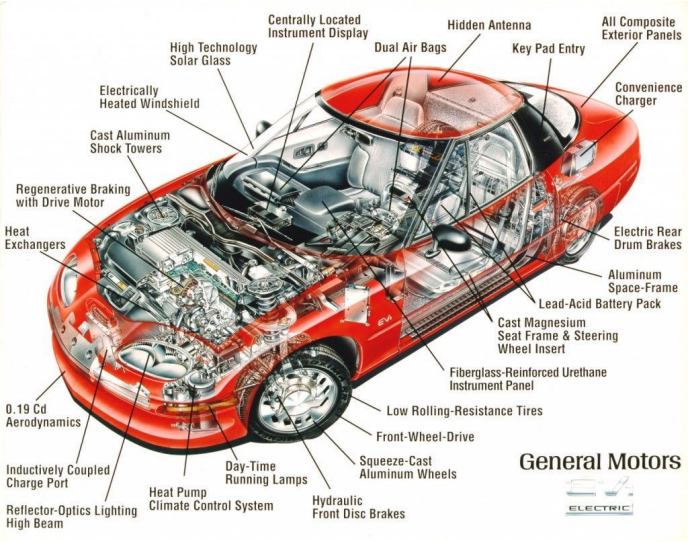


Your car, is a computer and a network!

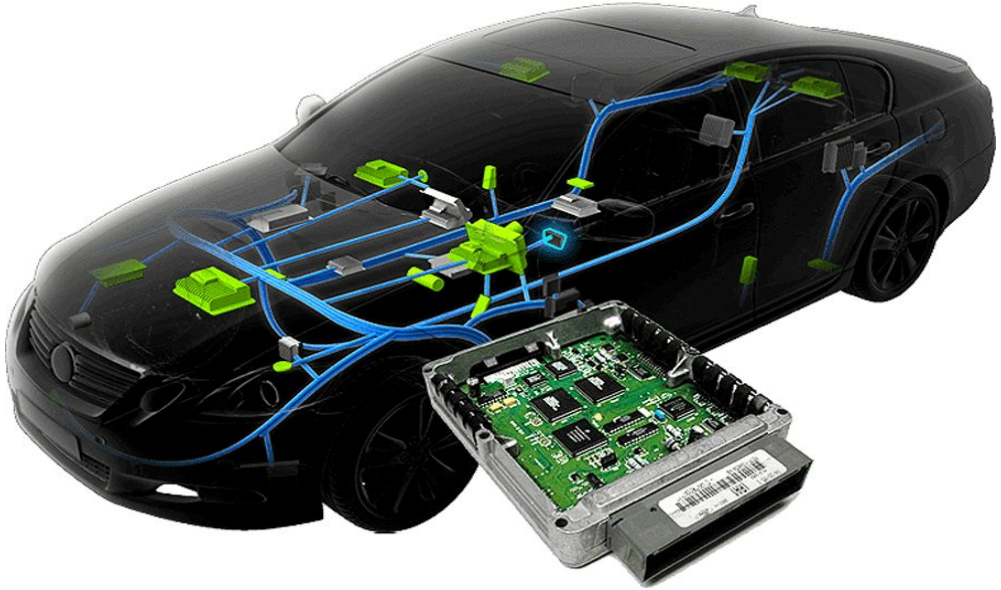


Your car, is a computer and a network!

A modern car can have as much as 50 ECU



Electronics Control Unit (ECU’s)



https://www.ccsentinel.com

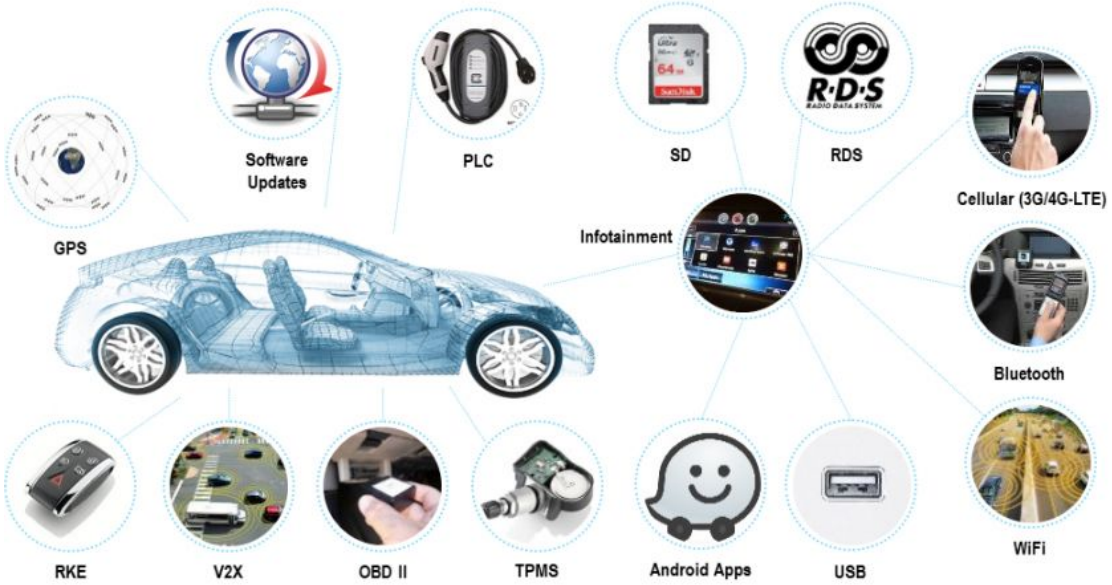
Identifying attack surface

Ask yourself these question, before identifying attack surface

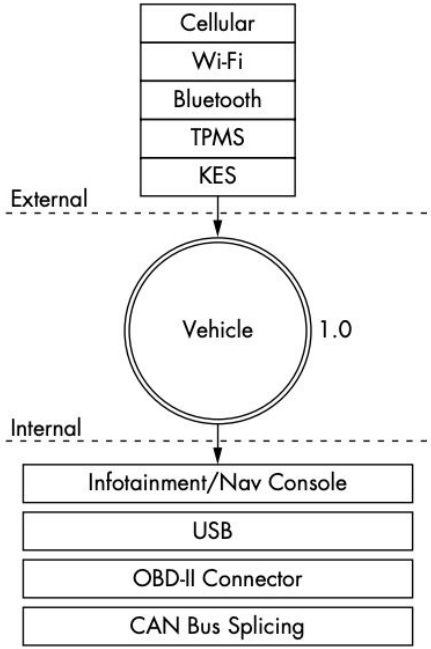
- Figure out the several signals received, Radio Waves, Key Fobs, Distance sensors etc.
- Is there a physical keypad?
- Any touch or motion sensor?
- Any diagnostic ports? OBD-II?
- Is there a infotainment system? Does it use bluetooth?

Find out several many ways that data can enter a vehicle. Question yourself, what if the data is malformed? Does it still function or it will stop responding or simply crash?

Attack surface on a modern vehicle



Attack surface on a modern vehicle - Bird’s Eye view



Network within the Car

- The CAN Bus
  - Released in 1986, mandatory from 2008
  - Runs on two wires: CAN high (CANH) and CAN low (CANL)
- The SAE J1850 Protocol
  - Developed in 1994
  - Older and slower than CAN
  - Much cheaper than CAN
- The Keyword Protocol
- The Local Interconnect Network Protocol
  - Cheapest among all
  - Complement to CAN
- The MOST Protocol
  - designed for multimedia services
- The FlexRay Bus
  - high-speed bus upto 10Mbps
  - Used for time sensitive communication
  - More expensive than CAN
- Automotive Ethernet
  - Cheaper alternative to MOST and FlexRay

The CAN bus

Controller Area Network

Released in 1986 by Bosch

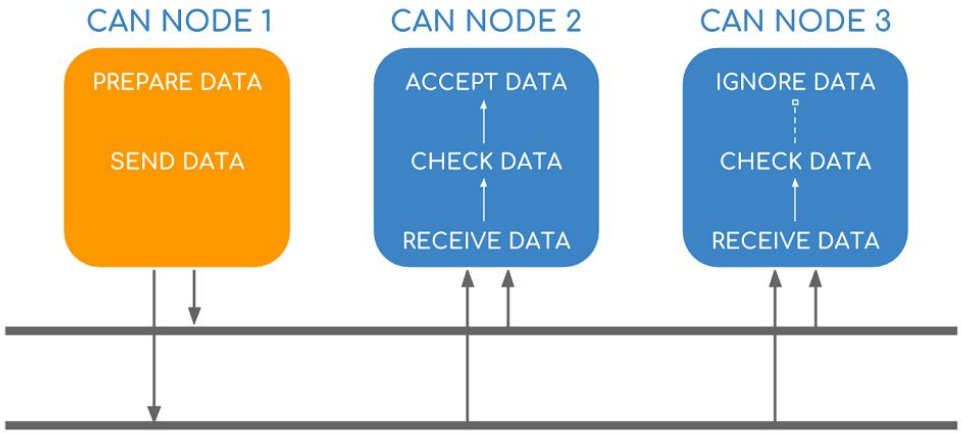
Central Nervous system that allows communication between all/some parts of a car

ISO 11898 defines CAN for Automotives

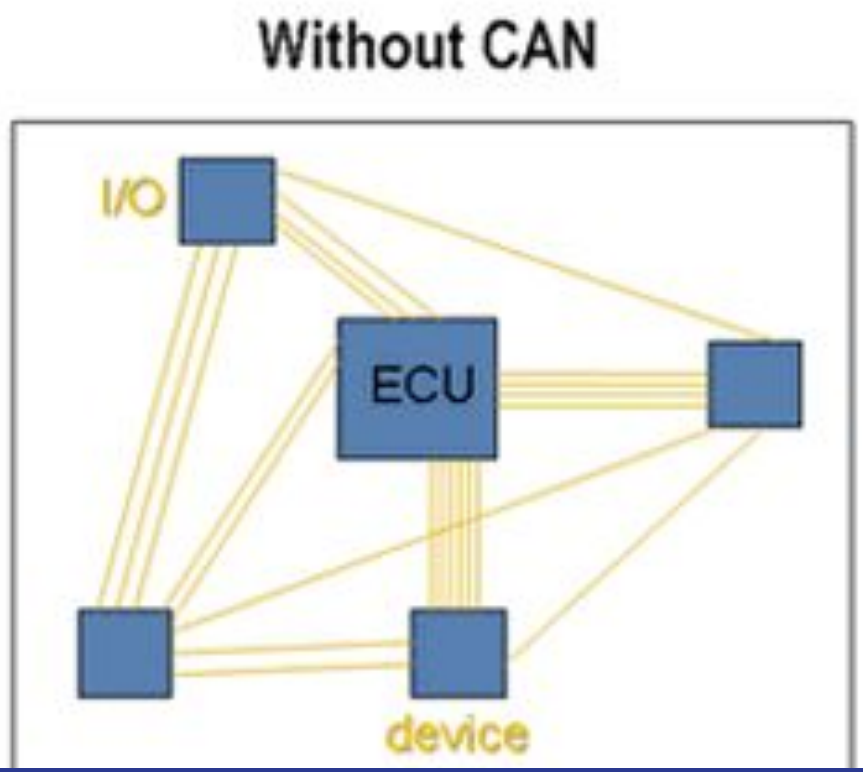
Runs on two different wires CANH and CANL

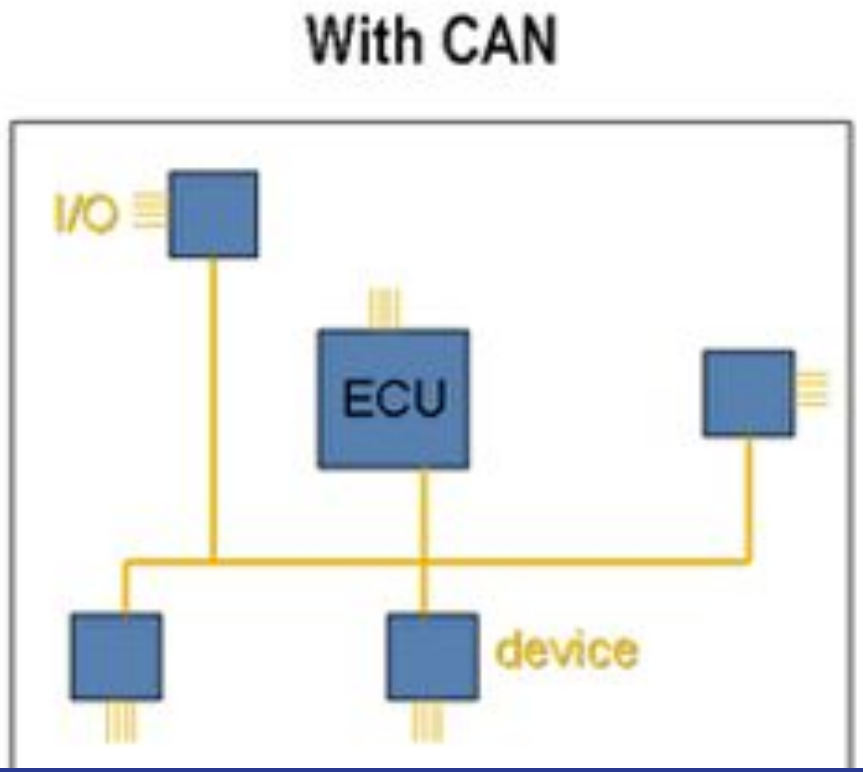
Every vehicle released after 2008 must have CAN

Typically more than 1 CAN bus on a modern Car
Tesla model S has 6 of them



With and Without CAN

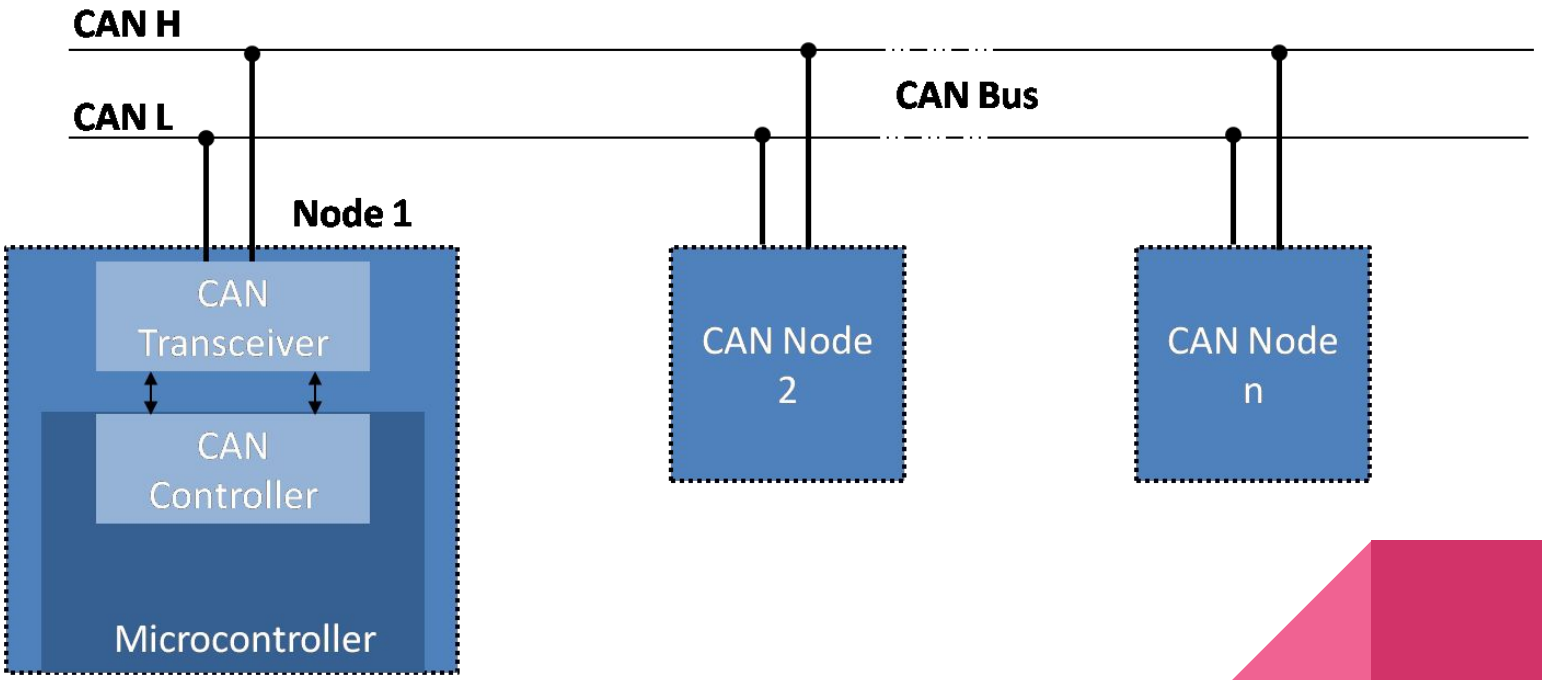




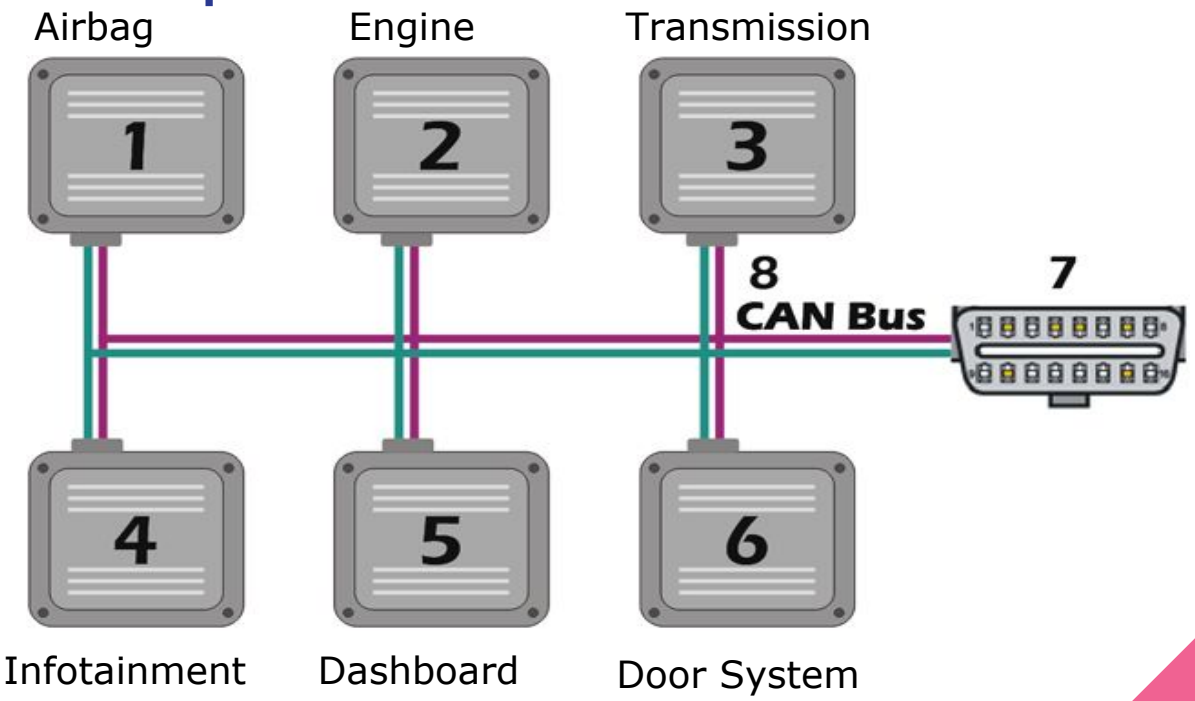
But, why CAN bus?

Really cheap to implement
Reliable
High Resilience to noise
Reduced wiring
Efficient
Mandated by Law

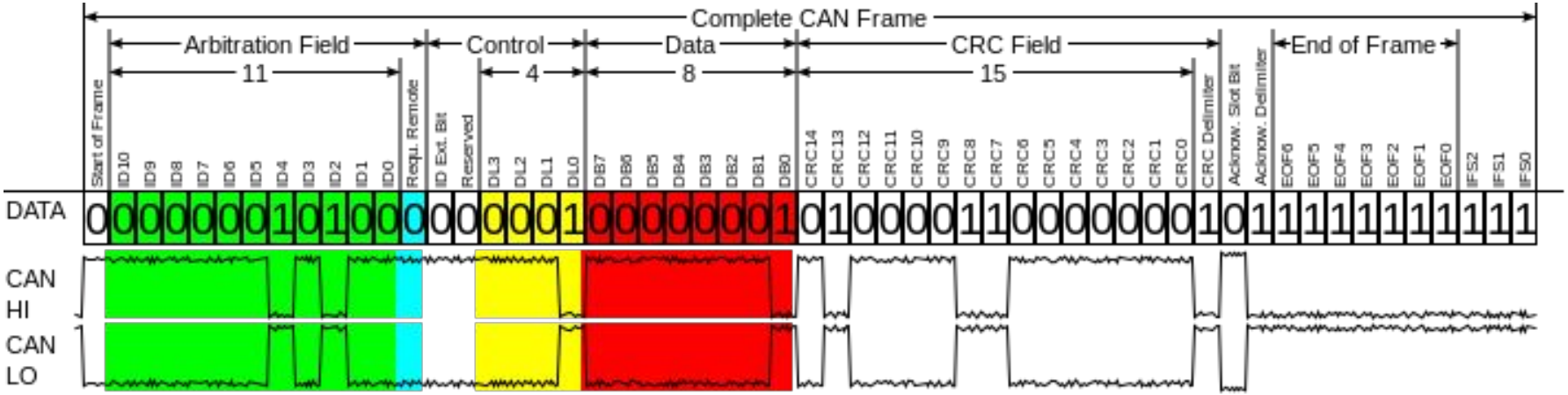
CAN Bus explained



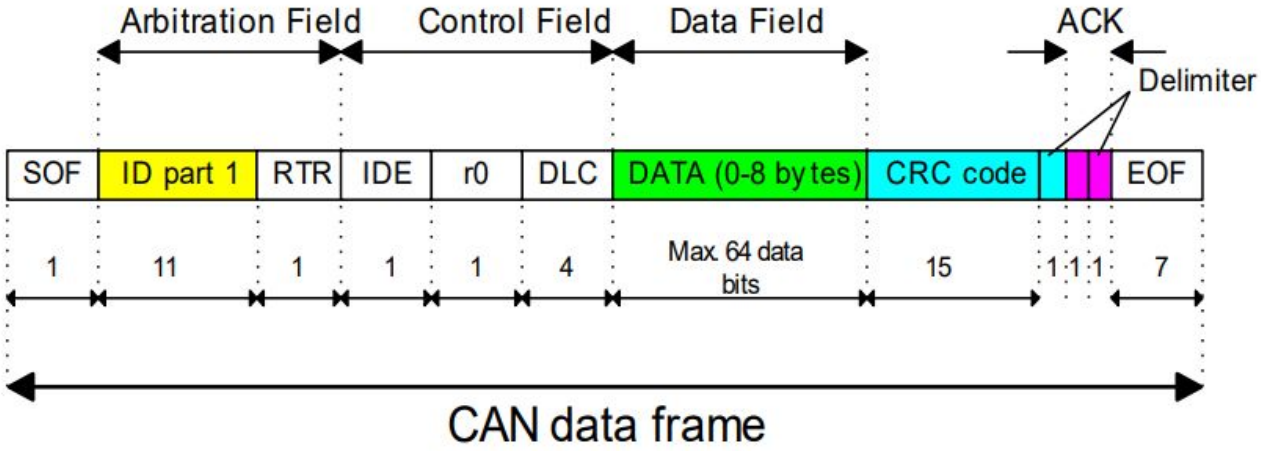
CAN Bus explained



CAN data frame



CAN data frame



CAN message Identifier

Lowest ID = Highest Priority

Airbag, ABS - Very High Priority, Lowest ID
Door Lock, Infotainment - Low Priority, Highest ID

CAN message structure

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x111	0x0B	0xB8	0x00	0x00	0x00	0x00	0x00	0XAB

CAN message structure

Engine Control Module

ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x111	0x0B	0xB8	0xED	0xAB	0xEF	0xEE	0xDC	0XAB

Engine RPM

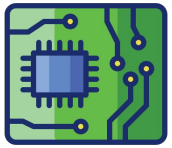
Instrument Cluster

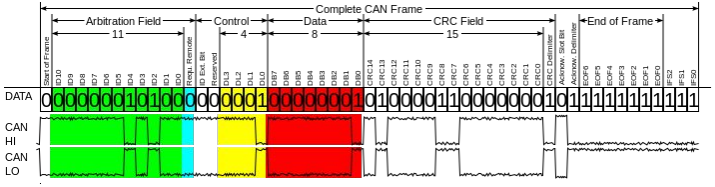
0x0BB8 = 3000

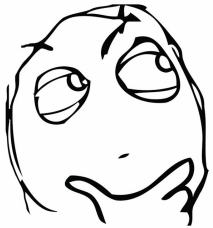
can-utils

```
vcan0  166   [4]  D0 32 00 18                '.2..'
vcan0  158   [8]  00 00 00 00 00 00 00 19    '........'
vcan0  161   [8]  00 00 05 50 01 08 00 1C    '...P....'
vcan0  191   [7]  01 00 90 A1 41 00 03       '....A..'
vcan0  133   [5]  00 00 00 00 A7             '.....'
vcan0  136   [8]  00 02 00 00 00 00 00 2A    '.......*'
vcan0  13A   [8]  00 00 00 00 00 00 00 28    '.......('
vcan0  13F   [8]  00 00 00 05 00 00 00 2E    '........'
vcan0  164   [8]  00 00 C0 1A A8 00 00 04    '........'
vcan0  17C   [8]  00 00 00 00 10 00 00 21    '.......!'
vcan0  18E   [3]  00 00 6B                   '..k'
vcan0  1CF   [6]  80 05 00 00 00 3C          '.....<'
vcan0  1DC   [4]  02 00 00 39                '...9'
vcan0  183   [8]  00 00 00 0E 00 00 10 2B    '.......+'
```

Journey so far...







What's Next?

How do I access the CAN Bus?

Journey so far...

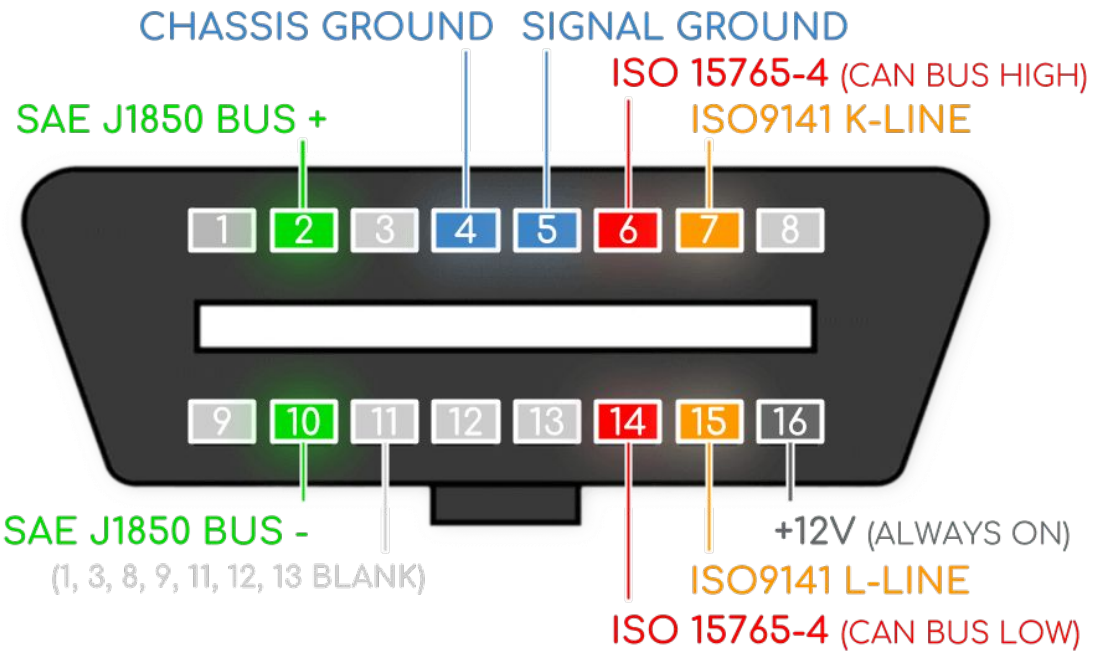
OBD-II Port

Locating the OBD Port

The OBD-II Port

- Found on vehicles after 1996
- Included in all modern cars
- Mandated by government for emission testing
- Direct access to CAN bus
- Standard Pinout, 6 & 14 for CAN, CAN High and CAN Low
- Direct communication on the CAN bus

The OBD-II Pinout



General methodology for CAN hacking

- Access to CAN Bus
- Sniff the packets
- Reverse Engineer the CAN packets
- Identify the Arbitration ID
- Replay!

Hardware/Software Needed

Hardware
- USB to CAN/CAN to USB

Software
- Read/Write CAN packets
- Encode/Decode CAN packets

OBD-II connectors - CAN Hardware

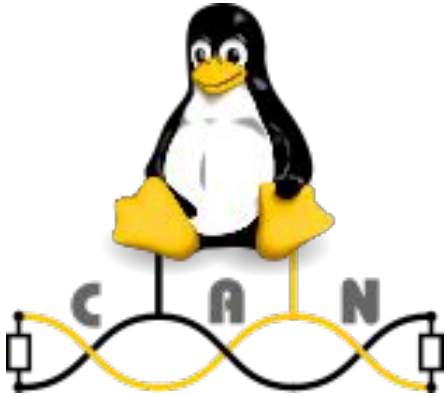
Hardware

- Kvaser $$$$
- EMS Wünsche $$$$
- Macchina M2 $$$
- Korlan USB2CAN $$
- ELM327, Terrible $

machina
M2
RESET
ERASE

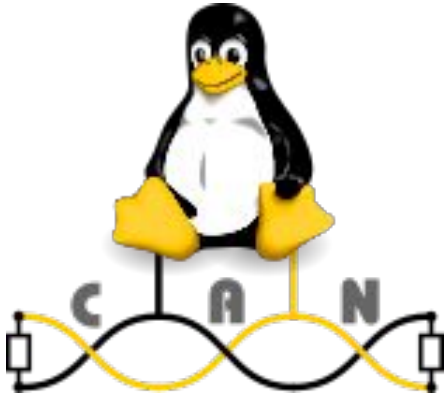
CAN Software

- SocketCAN, can-utils, vcan
- Wireshark
- SavvyCAN - My personal Favorite
- CANard
- carloop



SocketCAN

- CAN to LINUX/UNIX Network Interface
- Comes pre-packaged with Linux Kernel



can-utils

candump : display, filter and log CAN data to files

candump can0

canplayer : replay CAN log files

cansend : send a single frame

cangen : generate (random) CAN traffic

cansniffer : groups the CAN frames according to IDs

can-utils

```
vcan0  166   [4]  D0 32 00 18                '.2..'
vcan0  158   [8]  00 00 00 00 00 00 00 19    '........'
vcan0  161   [8]  00 00 05 50 01 08 00 1C    '...P....'
vcan0  191   [7]  01 00 90 A1 41 00 03       '....A..'
vcan0  133   [5]  00 00 00 00 A7             '.....'
vcan0  136   [8]  00 02 00 00 00 00 00 2A    '.......*'
vcan0  13A   [8]  00 00 00 00 00 00 00 28    '.......('
vcan0  13F   [8]  00 00 00 05 00 00 00 2E    '........'
vcan0  164   [8]  00 00 C0 1A A8 00 00 04    '........'
vcan0  17C   [8]  00 00 00 00 10 00 00 21    '.......!'
vcan0  18E   [3]  00 00 6B                   '..k'
vcan0  1CF   [6]  80 05 00 00 00 3C          '.....<'
vcan0  1DC   [4]  02 00 00 39                '...9'
vcan0  183   [8]  00 00 00 0E 00 00 10 2B    '.......+'
```

CAN spoofing on a Hyundai i10

Video

Demo #1

Agenda

- Installation of can utils, wireshark
- Playing around with canutils, vcan and wireshark

Can-utils installation

```
$ sudo apt-get install can-utils
```

Myth or Fact: Entry barrier for Car/CAN hacking is high

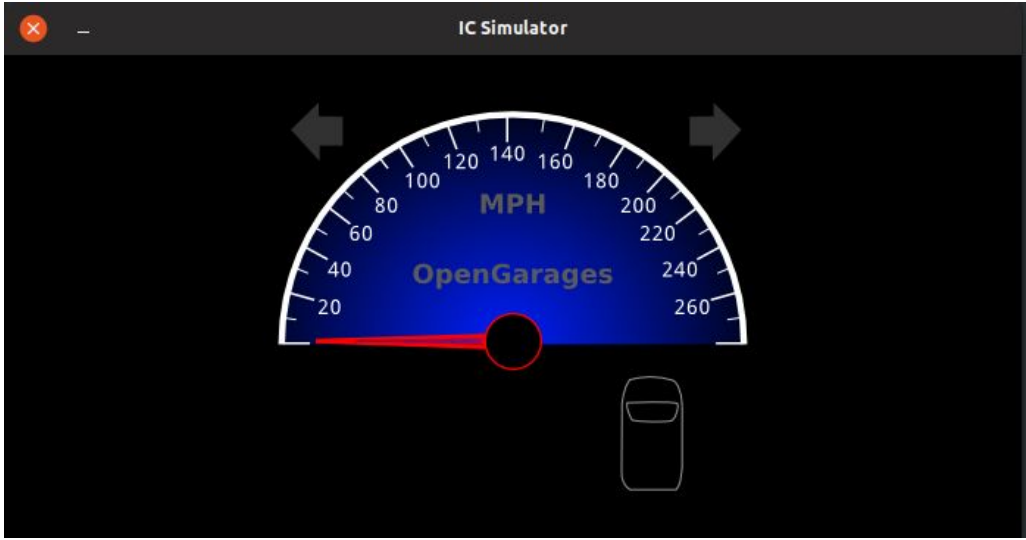
- Myth:
  - You would need to have a car to learn CAN hacking
  - You don’t even need to have a car to learn CAN hacking
- Myth:
  - You would need many expensive software toolkit
  - You have many open source tools to use for free.
- Fact:
  - You would need expensive hardware kit for CAN hacking.
  - Partly true, devices like USB2CAN can be pretty expensive sometimes.

ICSim: Instrument Cluster Simulator

By OpenGarages

Requires:
- SDL2
- SDL2_image
- can-utils

Open Source GUI Toolkit for Car Hacking
Created by Car Hacking researcher Craig Smith
Includes:
- Dashboard with speedometer
- Door lock
- Turn signal Indicators
- Control panel to interact with the simulated automobile network
    - Apply acceleration, brakes, control door locks, and turn signals



Demo #2 - Setting up the ICSim

Installing dependencies

```
$ sudo apt-get install libsdl2-dev libsdl2-image-dev
```

Install can-utils

```
$ sudo apt-get install can-utils
```

Download ICSim

```
$ git clone https://github.com/zombieCraig/ICSim.git
```

Setting up the virtual CAN interface

sudo modprobe can

sudo modprobe vcan

sudo ip link add dev vcan0 type vcan

sudo ip link set up vcan0

Verify the interface using ifconfig

Can-utils DEMO #3

Explanation

Candump - LOG

canplayer

cansend

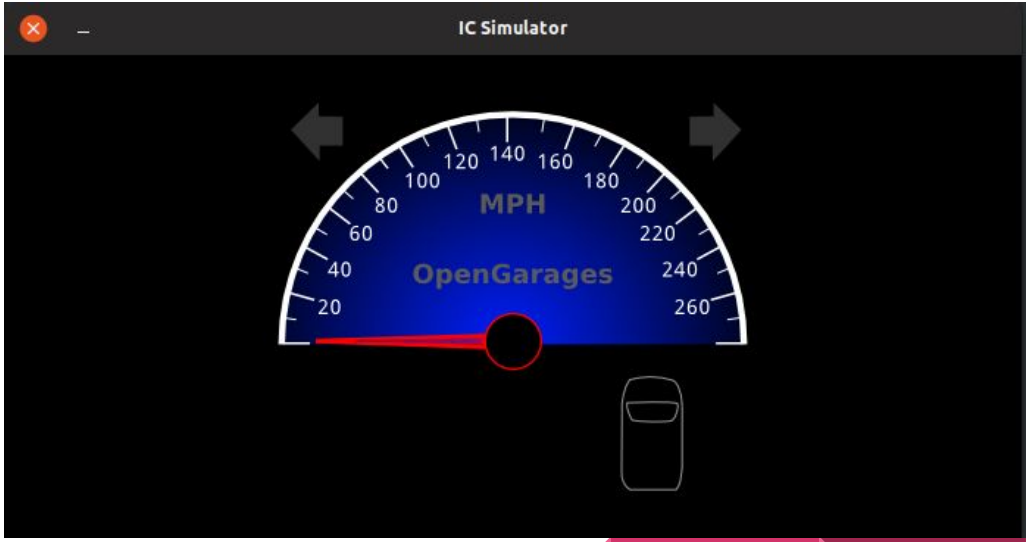
cangen

cansniffer

Demo #4 Introduction to ICSim

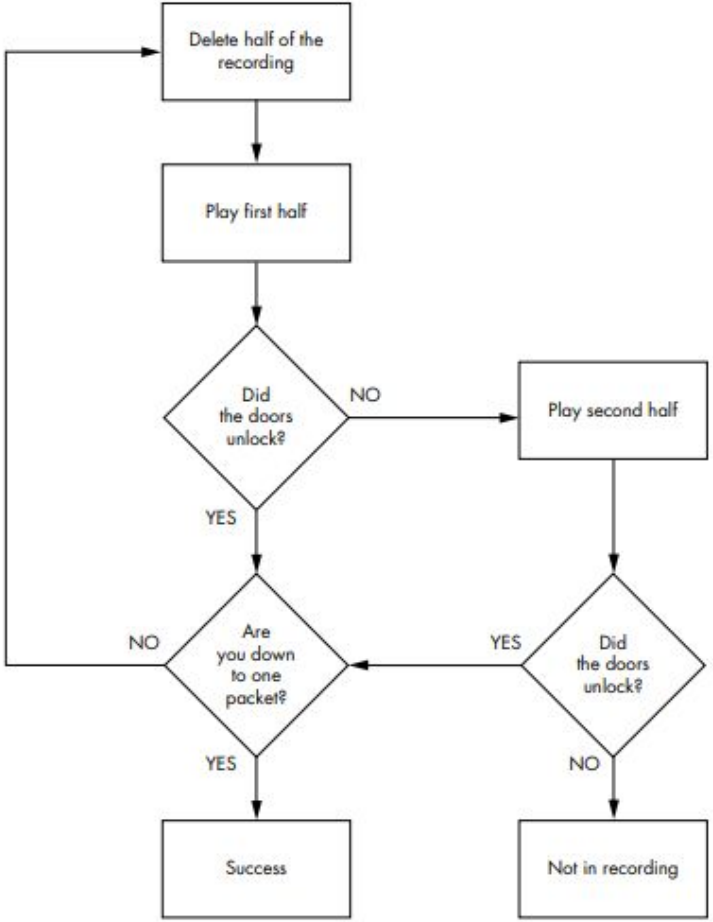
Agenda

- **Level 1**
- Introduction to ICSim
- Capturing CAN Frames
- Replay Attack
- Using cansend & cansniffer



Creative Packet Analysis

Use arbitration ID filtering Wisely



Smart Replay attack

Simple! Count the number of lines

```
$ wc -l whatever.log
```

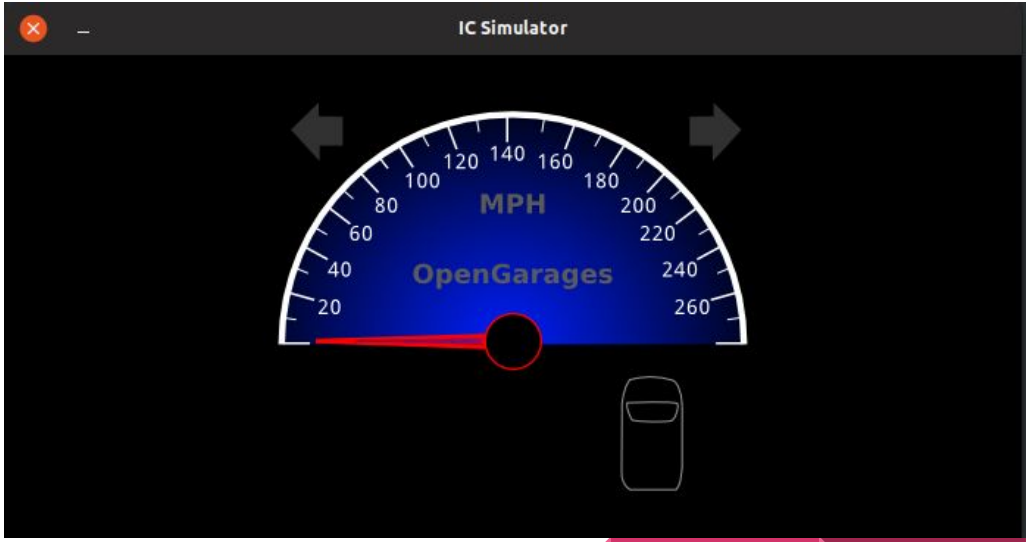
Split

```
$ split -l 5000 candump-whatever.log x1
```

Demo #4 Something more challenging

Agenda

- **Level 2**
- nano controls.c
- Change level to 2 line no 28
- $ make
- Re run the application



Challenge #1

- **Turn on the Hazard Light!**

Challenge #2

- **CAN you unlock the rear 2 doors together?**

SavvyCAN

SavvyCAN is a cross platform QT based C++ program. It is a CAN bus reverse engineering and capture tool. It was originally written to utilize EVTV hardware such as the EVTVDue and CANDue hardware. It has since expanded to be able to use any socketCAN compatible device as well as the Macchina M2 and Teensy 3.x boards. It can capture and send to multiple buses and CAN capture devices at once.

SavvyCAN is Awesome!!!

SavvyCAN File RE Tools Send Frames Connection

100% Fri 8:54 AM

Fuzzing Window

ng (ms)

a bytes

Burst Rate

1

Bus or Buses to Send On

All

ID Selection

Random

Range of IDs

Filter List

Bit Scanning

Sequential

Sweep

Random

All

None

BITS

7 6 5 4

0 1 2 3 4 5 6 7

s 0, Green = Fuzz Bit, Black = Bit always set, Gray = Past en

Fram

	En	Bus	ID	Len	Ext	Re
1						

Connection Settings

Connected Devices:

Type	Subtype	Port	Buses

Add New Device Connection

Reset Selected Device

Disconnect Selected Device

Bus Details:

Speed: 50000

Enable Console

Device Console:

Send:

Savvy CAN V199

Timestamp	ID	Ext	RTR	Dir	Bus	Len	ASCII	Data

Total Frames Captured:

0

Frames Per Second:

0

Suspend Capturing

Normalize Frame Timing

Clear Frames

Auto Scroll Window

Interpret Frames

Overwrite Mode

Frame Filtering:

Burst Rate:

Current frame:

nce

us

ID Filtering:

Installation of SavvyCAN

Download prebuilt binaries from https://www.savvycan.com/

If you wish to use SavvyCAN with ICSim, you need qt5

```
$ wget https://download.qt.io/official_releases/qt/5.12/5.12.4/qt-opensource-linux-x64-5.12.4.run

$ chmod a+x ./qt-opensource-linux-x64-5.12.4.run

$ sudo ./qt-opensource-linux-x64-5.12.4.run
```

qtserialbus is not included in the official apt repository in Ubuntu 18.04, so build it yourself.

```
$ sudo apt install qtdeclarative5-dev qttools5-dev g++

$ git clone https://github.com/qt/qtserialbus

$ cd qtserialbus

$ /opt/Qt5.12.4/5.12.4/gcc_64/bin/qmake .

$ make

$ sudo make install
```

Fuzzing with SavvyCAN

Scripting in SavvyCAN

One more reason to use SavvyCAN!

Thank you!

Car Hacker’s Handbook - Must read

More on can-utils & socketCAN, SavvyCAN

OpenGarages

Charlie Miller & Chris Valasek researches