workshop

November 7, 2020

1 Learn How to Hack a Car

1.0.1 A workshop by Amith Reddy

Please download the workshop materials here before proceeding: https://hackaday.io/project/175126-learn-how-to-hack-a-car-remoticon-workshop

Public chat also there. Its open to everyone you don't need to join team for access

2 Outline of this workshop

- Why hack cars?
- Difficulties in Car hacking

- Interactive Portion
  - 4 exercises
  - Would like volunteers for pacing
- Demos with python and SavvyCan
- Additional Resources

3 Who Am I?

- Recent mechatronics grad
- 100 hours into hobby. (Expert Beginner)
- I'm going to try to save you as much time and frustration as I can

4 Car hacking, why do it?

- to learn
- its fun
- to extend a cars functionality
- extends to other domains: heavy machinery, robotics
- repair (right to repair)

5 Obstacles

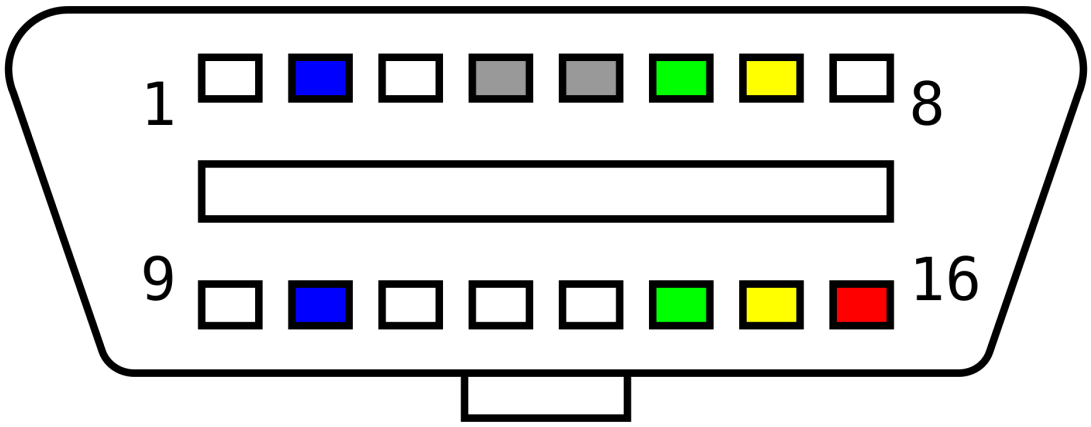
- Number and complexity of systems within a car

  - Wireless entry, Cellular connections, Satellite radio, GPS, Radar, ultrasonic, engine management, speakers, infotainment systems.
  - Auto brake
  - Lane keep systems
  - Advanced cruise control
- Require certain hardware to access systems (logic analyzers, SDR, etc)
- Differences between make and models
- Automakers want to hide the information to hinder repair.

That's why this workshop will use simulator ICSim to introduce beginners to car hacking

6 Easiest way to get started

- Simplest way to get started is at the OBD2 port.
- OBD2 port is a standard diagnostic port found on all cars built in the last 20 or so years. Its used for diagnostic purposes and emissions testing.



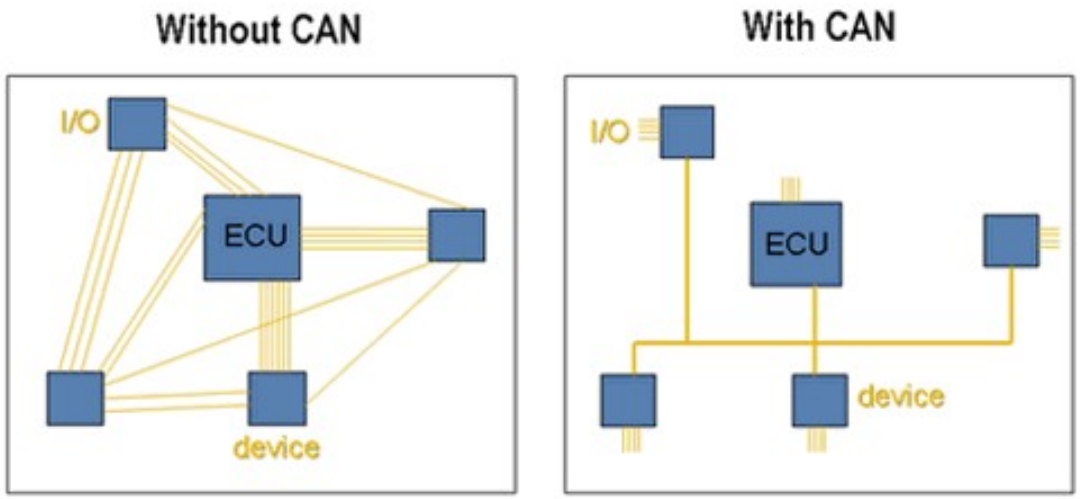
1	Manufacturer discretion. GM: J2411 GMLAN/SWC/Single-Wire CAN. VW/Audi: Switched +12 to tell a scan tool whether the ignition is on.	9	Manufacturer discretion. GM: 8192 baud ALDL where fitted. BMW: RPM signal.
2	Bus positive Line of SAE J1850 PWM and VPW	10	Bus negative Line of SAE J1850 PWM only (not SAE 1850 VPW)
3	Manufacturer discretion. Ford DCL(+) Argentina, Brazil (pre OBD-II) 1997–2000, USA, Europe, etc. Chrysler CCD Bus(+) Ethernet TX+ (Diagnostics over IP)	11	Manufacturer discretion. Ford DCL(-) Argentina, Brazil (pre OBD-II) 1997–2000, USA, Europe, etc. Chrysler CCD Bus(-) Ethernet TX- (Diagnostics over IP)
4	Chassis ground	12	Not connected Manufacturer discretion: Ethernet RX+ (Diagnostics over IP)
5	Signal ground	13	Manufacturer discretion. Ford: FEPS - Programming PCM voltage Ethernet RX- (Diagnostics over IP)
6	CAN high (ISO 15765-4 and SAE J2284)	14	CAN low (ISO 15765-4 and SAE J2284)
7	K-line of ISO 9141-2 and ISO 14230-4	15	L-line of ISO 9141-2 and ISO 14230-4
8	Manufacturer discretion. Many BMWs: A second K-line for non OBD-II (Body/Chassis /Infotainment) systems. Activate Ethernet (Diagnostics over IP)	16	Battery voltage

Image and Table Source

7 What is CAN? Why focus on it?

- Controller Area Network (CAN)
- All modern cars have it. Also heavy machinery and some robots.
- Omissions: hardware implementation. Low level details on CAN frames.

7.1 How does CAN work?



7.1.1 Who gets priority?

- The node with the lowest Arbitration ID

7.2 Benefits of CAN

- CAN has the highest bandwidth (expect ethernet)
- Very easy to add nodes to extend nodes
- Simple to implement physically

7.3 Limits of CAN

- Low bandwidth compared to automotive ethernet
- Max message length of 8 bytes
- Poor for Hard real time systems

8 Structure of a Can Packet

8.1 Parts of a message

- Arbitration ID 12 bits.
- Data up to 8 bytes

8.2 Example: 0x300 FF FF FF FF FF FF FF FF

- Where 0x300 is the ID

- Rest is the data

9 Exercise 0 Setup

Make sure you have download workshop files from my hackaday page.

- sudo modprobe can
- sudo modprobe vcan
- sudo ip link add dev vcan0 type vcan
- sudo ip link set up vcan0

Navigate to where you built icsim and run these commands - ./icsim vcan0 - ./controls vcan0

9.1 Controls for ICSIM

Make sure the controls window is selected when inputting commands. - Acceleration: up arrow - Blinkers: left, right arrow keys - Lock all doors: Hold Right Shift and tap Left Shift - Unlock all doors: Hold Left Shift and tap Right Shift

Test that you can lock and unlock doors

10 Exercise 1 Recording Signals while unlocking the car

Commands used for this part:

- candump -L > first.log vcan0
- less
- head -n 10 | log2long

11 Exercise 2 Finding the packet using replay attack

Commands we are going to use: - split -l number file prefix - split *file* into smaller files with up to *number* of lines and label new files using *Prefix* - wc -l file - counts the lines in a *file* - candump -L > first.log vcan0 - save can messages on vcan0 to *first.log* file - canplayer -I file - replay messages from *file*

12 Checking our work

- 19B is ID that locks and unlocks door
- run grep 19B on the file you narrowed down

Confirm using cansend - cansend vcan0 <id><message>

13 Exercise 3 Using cansniffer to find acceleration and blinkers

- Interactively find packets using cansniffer

Commands: * launch it with cansniffer -c vcan0 while running * # + enter - this will save bytes currently on screen * c + enter - this will toggle highlighting changing bytes

14 Demo: SavvyCan

- Repeat ex 2, 3 with SavvyCan

15 Demo Python

- Lets do ex 3 with python

15.1 Strategy

- Record ICSIM with candump in an idle state (no user input)
- Record ICSIM with candump while doing an action we are interested in
- Compare two log files to find similarities and differences.

[3]:
```
import pandas as pd
def file2dataframe(file):
    # Loads .log file. Also instructs pandas how to
    # parse the log file
    data_frame = pd.read_fwf(file,
                         delimiter=' \#',
                         delim_whitespace=True,
                         names=["time","interface","ID","Data"])
    sorted_df = data_frame.sort_values(by=["ID","time"]) # sort first by ID and␣
 ↪time
    sorted_df["Data"] = sorted_df["Data"].apply(convert)

    return sorted_df

def convert(x):
    return int(x,16)
def check_ID_trend(packetID, df):
    return df[ df['ID'] == packetID ]["Data"].plot
def packet_frequency(df):
    #returns a count of all unique CAN ID's
    keys= df["ID"].unique()
    total_counts= []
    unique_data=[]
    for key in keys:
        sel= df[df["ID"]== key]["Data"]
        unique_data.append(sel.unique().size)
        total_counts.append(sel.size)
    data = { "Counts":total_counts, "Unique": unique_data}
    count_df = pd.DataFrame( data, index=keys, columns=["Counts","Unique"])
    return count_df
```

[4]:
```
baseline_df = file2dataframe("icsim_baseline.log")
display(baseline_df)
```

```
                        time interface   ID                  Data
3       (1602977081.830363)     vcan0  039                    27
30      (1602977081.844925)     vcan0  039                    42
62      (1602977081.859039)     vcan0  039                    57
87      (1602977081.874258)     vcan0  039                    12
127     (1602977081.893879)     vcan0  039                    27
…                       …       …  …                   …
755030  (1602977466.796032)     vcan0  5A1  10808639105689215550
757497  (1602977468.054455)     vcan0  5A1  10808639105689215504
759459  (1602977469.054683)     vcan0  5A1  10808639105689215535
761422  (1602977470.055043)     vcan0  5A1  10808639105689215550
763891  (1602977471.314332)     vcan0  5A1  10808639105689215504

[765371 rows x 4 columns]
```

[5]:
```
base_unique_df = packet_frequency(baseline_df)
base_unique_df.sort_values(by=["Unique"])
display(base_unique_df.sort_values(by=["Unique"]))
pd.set_option('display.max_rows', 10)
```

```
     Counts  Unique
1D0   19395       1
188     775       1
5A1     359       3
1DC   19515       4
21E    9697       4
294    9697       4
305    3711       4
1CF   19514       4
309    3951       4
324    3951       4
333    3831       4
37C    3951       4
405    1317       4
40C    1317       4
428    1317       4
320    3951       4
1B0   19395       4
039   25741       4
1A4   19515       4
133   39029       4
136   39028       4
13A   39028       4
13F   39028       4
143   39029       4
158   39029       4
454    1317       4
161   39029       4
```

```
166   39029       4
17C   39028       4
18E   39028       4
191   39029       8
095   39029       8
164   39028       8
1AA   19395       8
183   39029      64
244   27359     255
```

[6]:
```
comp_df = file2dataframe("icsim_signals.log")

comp_unique_df = packet_frequency(comp_df)
comp_unique_df.sort_values(by=["Unique"])
```

[6]:
```
      Counts  Unique
1D0      410       1
5A1        8       3
188       17       3
1DC      411       4
21E      205       4
..       …     …
095      823       8
164      822       8
1AA      410       8
183      822      64
244      583     232

[36 rows x 2 columns]
```

[7]:
```
#Find IDs only in base_unique
diff_id=base_unique_df.index.difference(comp_unique_df.index)
display("Difference IDs",diff_id)

#Find IDs that exists in both dataframes
idx = comp_unique_df.index.intersection(base_unique_df.index)
base_unique_df.loc[idx]["Unique"].compare(comp_unique_df.loc[idx]["Unique"])
```

```
'Difference IDs'
```

```
Index([], dtype='object')
```

[7]:
```
       self  other
188     1.0    3.0
244   255.0  232.0
```

16 More complicated analysis possible

- Change in frequency
- Realtime frequency changes

17 Excercise 4 Canplayer and Cansniffer using real CAN traffic

- Files in CRV_2016_Touring
  - Idle.log
  - steering_wheel_buttons.log
  - steering_wheel_turning.log
  - wipers_and_stalks.log Playback using canplayer using specific command
- canplayer -I idle.log
- canplayer -I steering_wheel_turning.log

18 Demo finding a signal using Python and SavvyCan

18.1 Decoding a signal

If possible find somebody who already did this

19 DBC files

- What is a DBC file?
  - Describes the signals contained in message
- Where to get them
  - From other people
  - I got it from Comma Ai's OpenDBC project
    - https://github.com/commaai/opendbc
- SavvyCan demo with idle.log.
- idle.log file is a recording of a 2016 CRV with Touring trim sitting still in park

[8]:
```
filenames = ["idle.log","driving.log","steering_wheel_buttons.log",
             "steering_wheel_turning.log", "wipers_and_stalks.log"]

data_frames ={}

uniq_frames= {}

for name in filenames:
    # Create dataframes
    df= file2dataframe(name)
    # Store data frame and store it in dictionary
    data_frames[name]= df
    # Find Unique data and store it in dictionary
    uniq_frames[name] = packet_frequency(df)
```

[9]:
```
idledf= uniq_frames["idle.log"]
stalksdf= uniq_frames["wipers_and_stalks.log"]
drivingdf= uniq_frames["driving.log"]
```

[10]:
```
# Here I'm going to find the steering wheel input
steering_wheel = uniq_frames["steering_wheel_turning.log"]
idx = steering_wheel.index.intersection(idledf.index)
display("Same IDs",idx)
results = steering_wheel.loc[idx]["Unique"].compare(idledf.loc[idx]["Unique"])

pd.set_option('display.max_rows', 50)
results
```

```
'Same IDs'
```

```
Index(['094', '0C7', '0E7', '130', '14A', '158', '17C', '191', '1A4', '1A7',
       '1AB', '1B0', '1C2', '1D0', '1D5', '1D6', '1DA', '1DC', '1DF', '1ED',
       '1EF', '1FB', '309', '320', '324', '326', '32E', '331', '35E', '371',
       '372', '374', '378', '396', '39F', '403', '40F', '428', '444', '45B',
       '510', '516'],
      dtype='object')
```

[10]:

	self	other
094	30.0	73.0
0C7	22.0	28.0
0E7	54.0	40.0
130	2.0	4.0
14A	291.0	20.0
158	3.0	4.0
17C	89.0	158.0
191	2.0	4.0
1A4	4.0	10.0
1A7	2.0	4.0
1AB	201.0	14.0
1C2	2.0	4.0
1D5	2.0	4.0
1D6	4.0	3.0
1DA	2.0	4.0
1DC	139.0	122.0
1DF	2.0	4.0
1ED	2.0	4.0
1EF	2.0	4.0
1FB	2.0	4.0
309	2.0	4.0
320	2.0	4.0
324	9.0	139.0
326	4.0	13.0
32E	2.0	4.0

```
331    1.0    8.0
35E    2.0    4.0
371    2.0    4.0
372    2.0    4.0
374    2.0    6.0
378    2.0    3.0
396    2.0    8.0
39F    2.0    4.0
403    2.0    4.0
40F   11.0   28.0
45B    2.0    4.0
510    2.0    4.0
```

Consulting with the table above I choose packets that had more unique data in the left column which narrowed the results down to here: * 14A * 1AB

And looking at the datafile = "steering_wheel_turning.log" in SavvyCan I found that 14A ID contains the steering wheel angle

20 Demo More SavvyCan features:

20.0.1 - Range State

20.0.2 - Frame data analysis

20.0.3 - Flow view

21 Moving to the real world

- Safety
  - Careful working in enclosed spaces (CO poisoning)
  - Block wheels off
  - Avoid injecting CAN packets into a moving car
  - Elevate Car on jack stands
- Great hardware with great documentation will save you time
- CAN protocols you may run into
  - CAN FD extends length of data to 64 bytes.
  - ISOTOP
  - CANopen

22 Selecting hardware to connect to OBD port

What to look out for: - Cost - Documentation esp for car/automotive stuff - Form factor - Reading and writing to multiple buses simultaneously - What api work out of the box: socketcan, serial - logic analyzer may be a good place to start. Need to bring down the CAN voltage from 12V to 3.3V - A more comprehensive run down here: http://opengarages.org/handbook/ebook/#calibre_link-413

23 Resources

- Car hacker's handbook by Open garages. http://opengarages.org/handbook/ebook/
- Open garages email list: http://opengarages.org
- Comma ai: https://comma.ai/
  - github: https://github.com/commaai
  - openDBC https://github.com/commaai/opendbc
- Tools to develop CAN networks: https://github.com/GENIVI/CANdevStudio

24 Questions