Vehicle Communication with Virtual CAN Bus

Content

Abstract

This project consists of an advanced applications for open-source CAN tools, where I will perform filtering CAN data, logging CAN data and replaying CAN data. Vehicle CAN Buses are networks that connect controllers. These controllers are responsible for gathering and sharing information about sensors, vehicle states, information, and calibration. Each vehicle's network is expressly designed by the vehicle's manufacturer.

The data that goes across the networks is local to the vehicle. It is not meant to be routed or shared beyond the vehicle and its controllers (for the most part). So the data does not need to follow any protocol other than that of the physical layer itself. That's why when you connect to a vehicles CAN Bus you'll see a lot of data, but they'll be little to no indication on what the data means; what the content is in the data bytes.

This data is typically proprietary. It is there so controllers can share timely information with each other. This data is normally there (meaning you don't have to request it, it just shows up). That is why it is called Normal Mode Messages. If the vehicle's controllers are awake, they will transfer information between themselves. Many vehicles will have multiple CAN Buses and each network will have its own messages and information that needs to be shared between controllers. Sometimes the information will overlap with other networks, but often it does not.

Motivation

Understanding How Your Vehicle Works

The automotive industry has churned out some amazing vehicles, with complicated electronics and computer systems, but it has released little information about what makes those systems work. Once you understand how a vehicle's network works and how it communicates within its own system and outside of it, you'll be better able to diagnose and troubleshoot problems.

Validating the Security of Your Vehicle

As of this writing, vehicle safety guidelines don't address malicious electronic threats. While vehicles are susceptible to the same malware as your desktop, automakers aren't required to audit the security of a vehicle's electronics. This situation is simply unacceptable: we drive our families and friends around in these vehicles, and every one of us needs to know that our vehicles are as safe as can be. If you learn how to hack your car, you'll know where your vehicle is vulnerable so that you can take precautions and be a better advocate for higher safety standards.

Chapter 1

Introduction

CAN Bus :

A Controller Area Network (CAN bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but can also be used in many other contexts.

Differential signals :

CANH is the HIGH-level CAN bus line which is a differential signal. CANL is the LOW-level CAN bus line which is a differential signal. There are two logical states: dominant and recessive. The figure below shows the general concept :

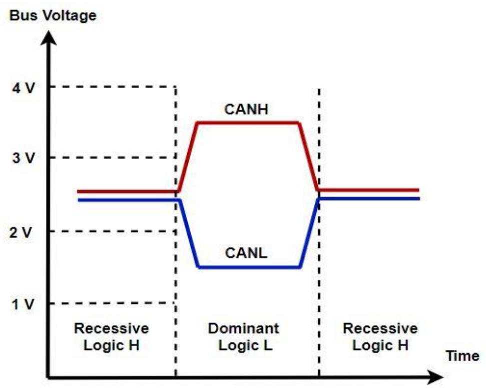


Fig. 1.1

- Logic 1 – Recessive State. Both CAN H and CAN L are at 2.5V.
- Logic 0 – Dominant state. CAN H is 3.5V and CAN L is 1.5V.
- The difference between CAN H and CAN L is 2V, which is dominant.
- There is a spike in voltage because of an electromagnetic interference, noise.

Standard CAN Frame Formats:

The data frame is the only frame for actual data transmission. There are two message formats:

- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

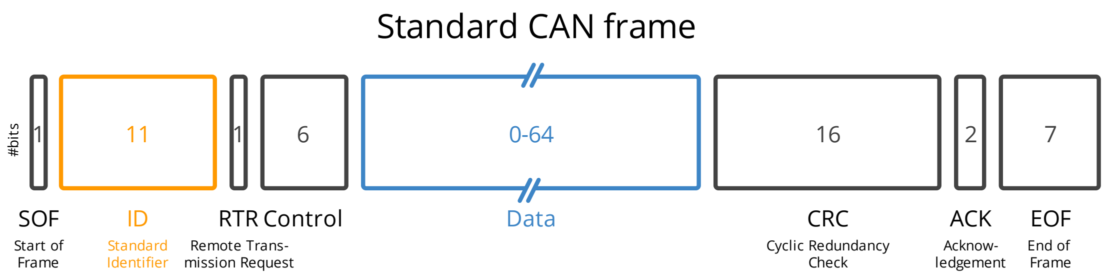


Fig. 1.2

CAN Arbitration:

The arbitration ID is a broadcast message that identifies the ID of the device trying to communicate, though any one device can send multiple arbitration IDs. If two CAN packets are sent along the bus at the same time, the one with the lower arbitration ID wins.

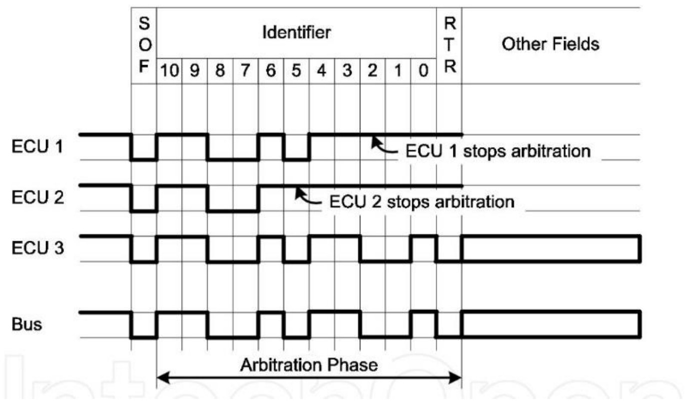


Fig. 1.3

• **SOF** : The single dominant start of frame (SOF) bit marks the start of a message and is used to synchronize the nodes on a bus after being idle.

• **Identifier** : The Standard CAN 11-bit identifier establishes the priority of the message. The lower the binary value, the higher its priority.

• **RTR** : The single remote transmission request (RTR) bit is dominant when information is required from another node. All nodes receive the request, but the identifier determines the specified node.

• **IDE** : A dominant single identifier extension (IDE) bit means that a standard CAN identifier with no extension is being transmitted.

• **r0** : Reserved bit (for possible use by future standard amendment).

• **DLC** : The 4-bit data length code (DLC) contains the number of bytes of data being transmitted.

• **Data** : Up to 64 bits of application data may be transmitted.

• **CRC** : The 16-bit (15 bits plus delimiter) cyclic redundancy check (CRC) contains the checksum (number of bits transmitted) of the preceding application data for error detection.

• **ACK** : Every node receiving an accurate message overwrites this recessive bit in the original message with a dominate bit, indicating an error-free message has been sent. Should a receiving node detect an error and leave this bit recessive, it discards the message and the sending node repeats the message after rearbitration. In this way, each node acknowledges (ACK) the integrity of its data. ACK is 2 bits, one is the acknowledgment bit and the second is a delimiter.

• **EOF** : This end-of-frame (EOF), 7-bit field marks the end of a CAN frame (message) and disables bitstuffing, indicating a stuffing error when dominant. When 5 bits of the same logic level occur in succession during normal operation, a bit of the opposite logic level is stuffed into the data.

• **IFS** : This 7-bit interframe space (IFS) contains the time required by the controller to move a correctly received frame to its proper position in a message buffer area.

Chapter 2

Vehicle Communication with SocketCAN

Volkswagen Group Research contributed the original Socket CAN implementation, which supports built-in CAN chips and card drivers, external USB and serial CAN devices, and virtual CAN devices. The can-utils package provides several applications and tools to interact with the CAN network devices, CAN-specific protocols, and the ability to set up a virtual CAN environment.

Setting Up can-utils to Connect to CAN Devices :

Prerequisite:
GNU/Linux platform(Kali Linux)
We decided to write the tool in the first place for Linux operating systems, because of their native support of SocketCAN.

Installing can-utils:

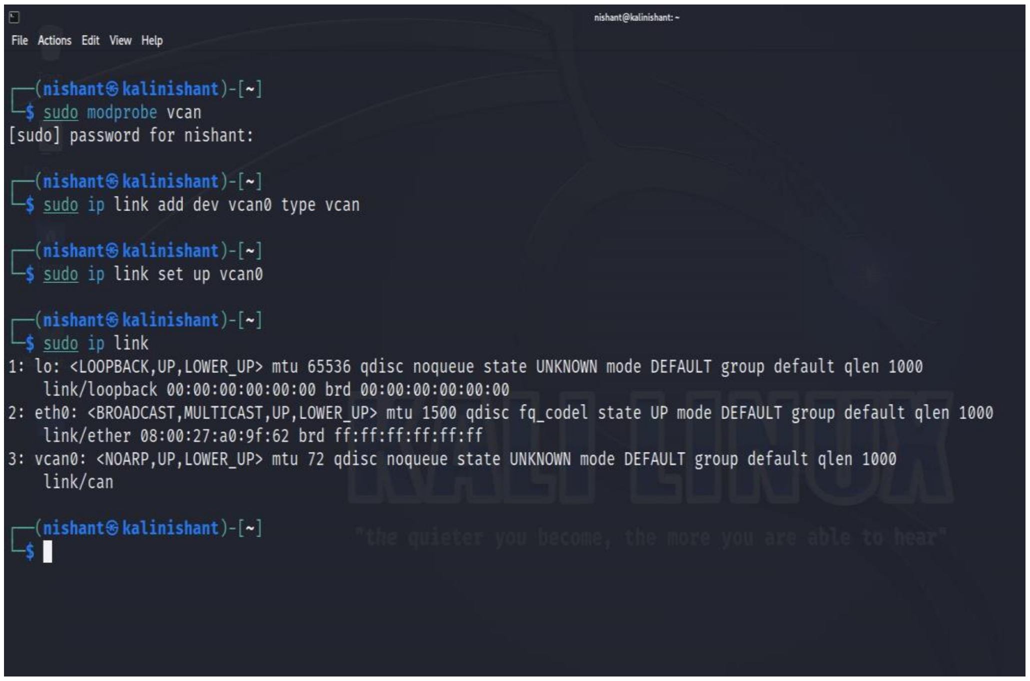
$ sudo apt-get install can-utils

Setting Up a Virtual CAN Network:

$ modprobe vcan (vcan is a user selected network device we created)

$ ip link add dev vcan0 type vcan

$ ip link set up vcan0



```
┌──(nishant㉿kalinishant)-[~]
└─$ sudo modprobe vcan
[sudo] password for nishant:

┌──(nishant㉿kalinishant)-[~]
└─$ sudo ip link add dev vcan0 type vcan

┌──(nishant㉿kalinishant)-[~]
└─$ sudo ip link set up vcan0

┌──(nishant㉿kalinishant)-[~]
└─$ sudo ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:a0:9f:62 brd ff:ff:ff:ff:ff:ff
3: vcan0: <NOARP,UP,LOWER_UP> mtu 72 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/can

┌──(nishant㉿kalinishant)-[~]
└─$ 
```

The CAN Utilities Suite

cansend : This tool sends a single CAN frame to the network.

Syntax : cansend<device><can_id>#{data}

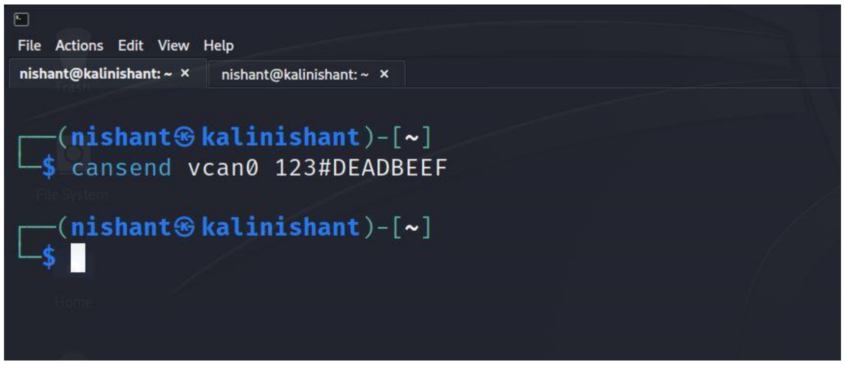
Options:

<can_id>#{data}
for 'classic' CAN 2.0 data frames
<can_id>#R{len}
for 'classic' CAN 2.0 data frames
<can_id>: 8 (EFF) hex chars

{data}: 0..8 bytes of hex-values for CAN 2.0.
{len}: an optional 0..8 value as RTR frames can contain a valid dlc field.

Examples:

- $ cansend vcan0 5A1#11.2233.44556677.88
- $ cansend vcan0 123#DEADBEEF



```
┌──(nishant㉿kalinishant)-[~]
└─$ cansend vcan0 123#DEADBEEF

┌──(nishant㉿kalinishant)-[~]
└─$ 
```

candump: This utility dumps CAN packets. It can also take filters and log packets.

Syntax : candump [options] <CAN interface>+

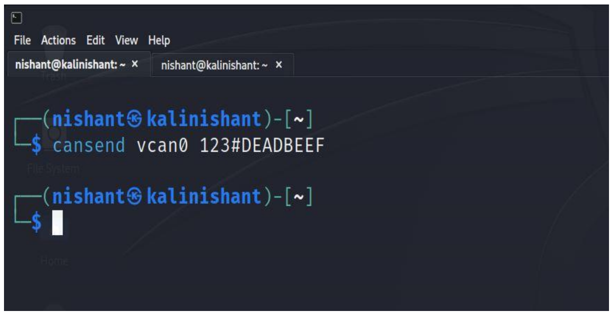
Options:

-t <type> (timestamp: (a)bsolute/(d)elta/(z)ero/(A)bsolute w date)
-c (increment color mode level)
-l (log CAN-frames into file. Sets '-s 2' by default)
-L (use log file format on stdout)
-n <count> (terminate after reception of <count> CAN frames)

Examples:

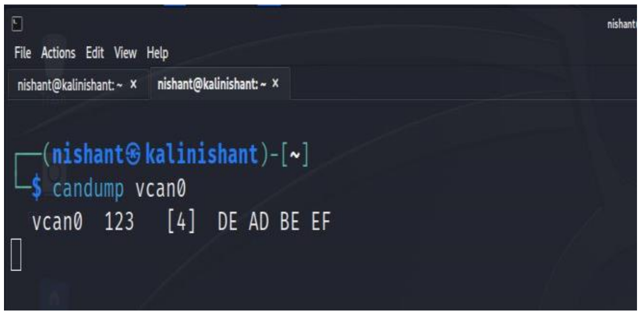
- $ candump vcan0
- $ candump -c vcan0, 4EF:7FF

We can see the first column tells us what interface we are on. The second column tells us what Arbitraion ID (ArbID). This ID is to note the format of the data bytes. Lastly we see the data bytes. CAN Frames cannot exceed 8 bytes of data. (Press CNTL+C to escape)



```
┌──(nishant㉿kalinishant)-[~]
└─$ cansend vcan0 123#DEADBEEF

┌──(nishant㉿kalinishant)-[~]
└─$ 
```



```
┌──(nishant㉿kalinishant)-[~]
└─$ candump vcan0
  vcan0  123   [4]  DE AD BE EF
```

We can see in (terminal 1) we are sending can data using cansend and we are receiving data using candump(terimanl 2). We are working in the terminal we need to have two terminal sessions running. One for sending and the other for receiving. This way when we send a message, we can see that it sent on our receive terminal.

Filtering CAN data using candump Filters:
Filtering method is used for filtering out a specific data that we want from random generated data from CAN bus. It is a simple way to allow you to see only certain frame ArbIDs.

Filtering has two forms:
[i] Inclusion filter:
<can_id>:<can_mask>
(matches when <received_can_id> & mask == can_id & mask)

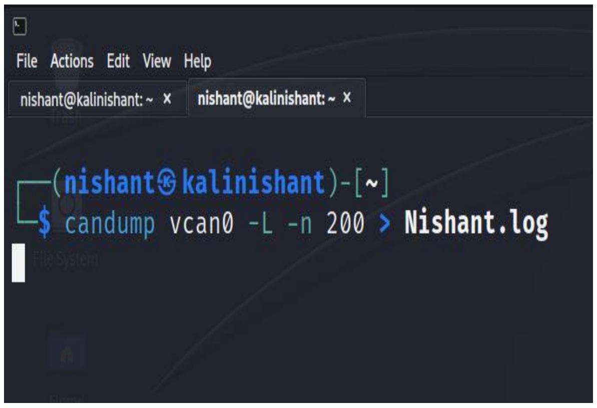
[ii] Exclusion filter:
<can_id>~<can_mask>
(matches when <received_can_id> & mask != can_id & mask)
#<error_mask>

Mask : It is a string of bits. It is used to be bits ‘1’ or ‘0’. It lets you control multiple bits with one operation. It uses only AND logic operation in candump.

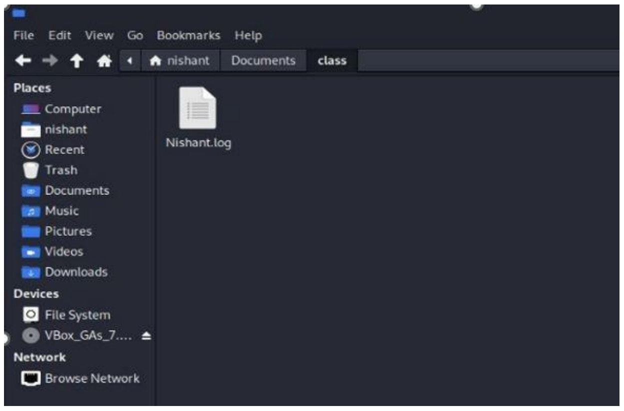
Examples : 7FF – [1 1 1 1 1 1 1 1 1 1 1]
7FE – [1 1 1 1 1 1 1 1 1 1 0]

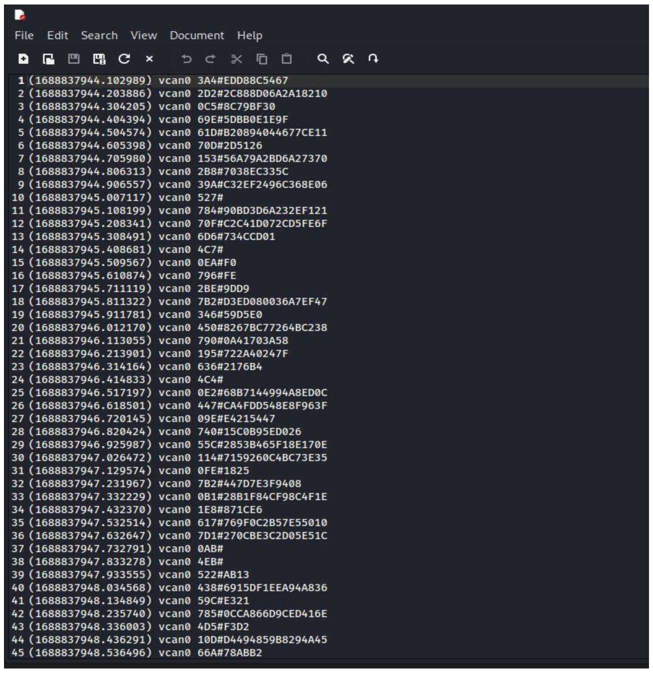
Logging with candump: It lets you customize log file name.

Syntax : $candump vcan0 -L > FileName.log



```
┌──(nishant㉿kalinishant)-[~]
└─$ candump vcan0 -L -n 200 > Nishant.log
```





```
(1688837944.102989) vcan0 3A4#EDD88C5467
(1688837944.203886) vcan0 2D2#2C888D06A2A18210
(1688837944.304205) vcan0 0C5#8C79BF30
(1688837944.404394) vcan0 69E#5DBB0E1E9F
(1688837944.504574) vcan0 61D#B20894044677CE11
(1688837944.605398) vcan0 70D#2D5126
(1688837944.705980) vcan0 153#56A79A2BD6A27370
(1688837944.806313) vcan0 2B8#7038EC335C
(1688837944.906557) vcan0 39A#C32EF2496C368E06
(1688837945.007117) vcan0 527#
(1688837945.108199) vcan0 784#90BD3D6A232EF121
(1688837945.208341) vcan0 70F#C2C41D072CD5FE6F
(1688837945.308491) vcan0 6D6#734CCD01
(1688837945.408681) vcan0 4C7#
(1688837945.509567) vcan0 0EA#F0
(1688837945.610874) vcan0 796#FE
(1688837945.711119) vcan0 2BE#9DD9
(1688837945.811322) vcan0 7B2#D3ED080036A7EF47
(1688837945.911781) vcan0 346#59D5E0
(1688837946.012170) vcan0 450#8267BC77264BC238
(1688837946.113055) vcan0 790#0A41703A58
(1688837946.213901) vcan0 195#722A40247F
(1688837946.314164) vcan0 636#2176B4
(1688837946.414833) vcan0 4C4#
(1688837946.517197) vcan0 0E2#68B7144994A8ED0C
(1688837946.618501) vcan0 447#CA4FDD548E8F963F
(1688837946.720145) vcan0 09E#E4215447
(1688837946.820424) vcan0 740#15C0B95ED026
(1688837946.925987) vcan0 55C#2853B465F18E170E
(1688837947.026472) vcan0 114#7159260C4BC73E35
(1688837947.129574) vcan0 0FE#1825
(1688837947.231967) vcan0 7B2#447D7E3F9408
(1688837947.332229) vcan0 0B1#28B1F84CF98C4F1E
(1688837947.432370) vcan0 1E8#871CE6
(1688837947.532514) vcan0 617#769F0C2B57E55010
(1688837947.632647) vcan0 7D1#270CBE3C2D05E51C
(1688837947.732791) vcan0 0AB#
(1688837947.833278) vcan0 4EB#
(1688837947.933555) vcan0 522#AB13
(1688837948.034568) vcan0 438#6915DF1EEA94A836
(1688837948.134849) vcan0 59C#E321
(1688837948.235740) vcan0 785#0CCA866D9CED416E
(1688837948.336003) vcan0 4D5#F3D2
(1688837948.436291) vcan0 10D#D4494859B8294A45
(1688837948.536496) vcan0 66A#78ABB2
```

Playing back Log files :

- Replay attack
- Digital forensic
- Driving simulation
- Product demos

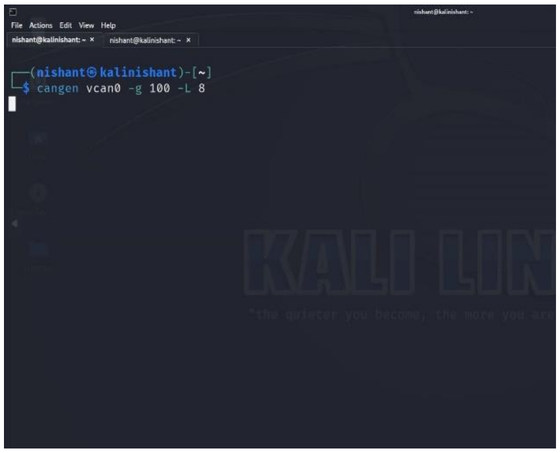
cangen : This tool used to generate large volumes of CAN data. It is good for testing purposes.
By default, it generates randomized CAN data on the bus.

Syntax : cangen [options] <CAN interface>

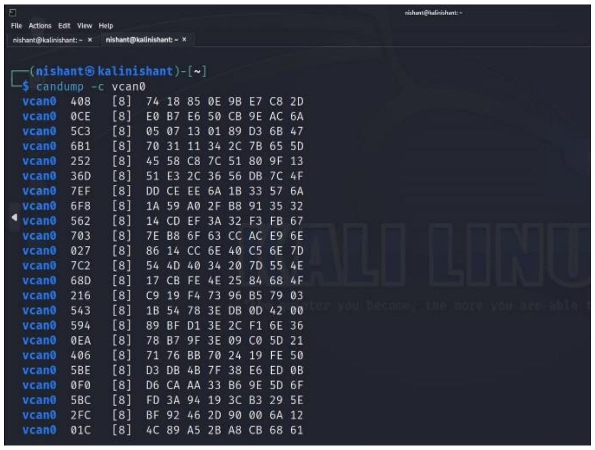
Options:
-g <ms> (gap in milli seconds - default: 200 ms)
-I <mode> (CAN ID generation mode - see below)
-L <mode> (CAN data length code (dlc) generation mode - see below)
-D <mode> (CAN data (payload) generation mode - see below)
-n <count> (terminate after <count> CAN frames - default infinite)

Generation modes: When incrementing the CAN data the data length code minimum is set to 1. CAN IDs and data content are given and expected in hexadecimal values.

'r'	=> random values (default)
'i'	=> increment values
<hexvalue>	=> fix value using <hexvalue>



```
┌──(nishant㉿kalinishant)-[~]
└─$ cangen vcan0 -g 100 -L 8
```



```
┌──(nishant㉿kalinishant)-[~]
└─$ candump -c vcan0
  vcan0  408   [8]  74 18 85 0E 9B E7 C8 2D
  vcan0  0CE   [8]  E0 B7 E6 50 CB 9E AC 6A
  vcan0  5C3   [8]  05 07 13 01 89 D3 6B 47
  vcan0  6B1   [8]  70 31 11 34 2C 7B 65 5D
  vcan0  252   [8]  45 58 C8 7C 51 80 9F 13
  vcan0  36D   [8]  51 E3 2C 36 56 DB 7C 4F
  vcan0  7EF   [8]  DD CE EE 6A 1B 33 57 6A
  vcan0  6F8   [8]  1A 59 A0 2F B8 91 35 32
  vcan0  562   [8]  14 CD EF 3A 32 F3 FB 67
  vcan0  703   [8]  7E B8 6F 63 CC AC E9 6E
  vcan0  027   [8]  86 14 CC 6E 40 C5 6E 7D
  vcan0  7C2   [8]  54 4D 40 34 20 7D 55 4E
  vcan0  68D   [8]  17 CB FE 4E 25 84 68 4F
  vcan0  216   [8]  C9 19 F4 73 96 B5 79 03
  vcan0  543   [8]  1B 54 78 3E DB 0D 42 00
  vcan0  594   [8]  89 BF D1 3E 2C F1 6E 36
  vcan0  0EA   [8]  78 B7 9F 3E 09 C0 5D 21
  vcan0  406   [8]  71 76 BB 70 24 19 FE 50
  vcan0  5BE   [8]  D3 DB 4B 7F 38 E6 ED 0B
  vcan0  0F0   [8]  D6 CA AA 33 B6 9E 5D 6F
  vcan0  5BC   [8]  FD 3A 94 19 3C B3 29 5E
  vcan0  2FC   [8]  BF 92 46 2D 90 00 6A 12
  vcan0  01C   [8]  4C 89 A5 2B A8 CB 68 61
```

You will start to see that some ArbID's are the same with the data sometimes changing other times you could have the same ID and different data. Normal Mode messages are the data exchanged normally between controllers. These messages are proprietary, so we don't know what they mean. By reverse engineering the data we can understand what the bits and bytes mean.

canplayer : It plays back candump files.

Syntax : canplayer <options> [interface assignment]*

Options :
-I <Infile> (default stdin)
-l <num> (process input file <num> times) (Use 'i' for infinite loop - default: 1)
-t (ignore timestamps: send frames immediately)
-g <ms> (gap in milli seconds - default: 1 ms)

Example : canplayer -l Nishant.log

Interface assignments:

- 0..n assignments like <write-if>=<log-if>
- e.g. vcan2=can0 (send frames received from can0 on vcan2)
- No assignments => send frames to the interface(s) they had been received from

cansniffer : This tool groups the packets by arbitration ID and highlights the bytes that have changed since the last time the sniffer looked at that id.

Syntax : cansniffer [can-interface]

Options :
-b (start with binary mode)
-8 (start with binary mode - for EFF on 80 chars)
-B (start with binary mode with gap - exceeds 80 chars!)
-c (color changes)
-t <time> (timeout for ID display [x10ms] default: 500, 0 = OFF)

Chapter 3

Testing CAN network

Instrument Cluster Simulator (ICSim) :

ICSim is a software utility designed to produce a few key CAN signals in order to provide a lot of seemingly "normal" background CAN noise—essentially, it's designed to let you practice CAN bus reversing without having to tinker around with your car. (ICSim is Linux only because it relies on the virtual CAN devices). ICSim will directly translate to your target vehicles. ICSim was designed as a safe way to familiarize yourself with CAN reversing so that the transition to an actual vehicle is as seamless as possible.

Setting Up the ICSim : ICSim comes with two components, icsim and controls, which talk to each other over a CAN bus. To use ICSim, first load the instrument cluster to the vcan device like this:

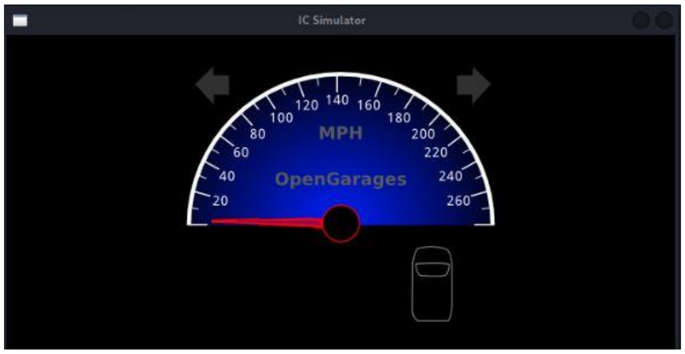
Installing ICSim:

sudo apt-get install libsdl2-dev libsdl2-image-dev –y

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

~/ICSim /icsim vcan0

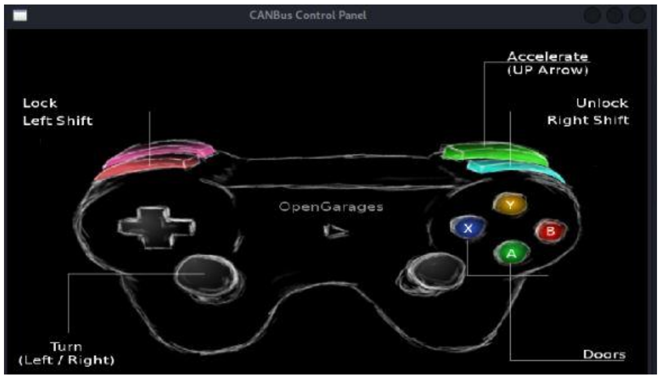
In response, you should see the ICSim instrument cluster with turn signals, a speedometer, and a picture of a car, which will be used to show the car doors locking and unlocking.



The icsim application listens only for CAN signals, so when the ICSim first loads, you shouldn't see any activity. In order to control the simulator, load the CANBus Control Panel like this:

~/ICSim/controls vcan0

The CAN Bus Control Panel shown below :



The screen looks like a game controller; in fact, you can plug in a USB game controller, and it should be supported by ICSim. You can use the controller to operate the ICSim in a method similar to driving a car using a gaming console, or you can control it by pressing the corresponding keys on your keyboard (see Figure 5-9).

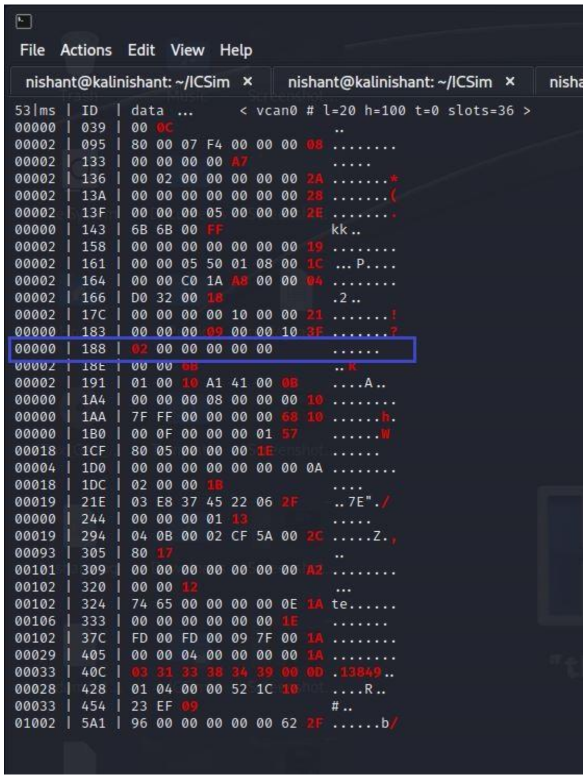
The main controls on the CANBus Control Panel are as follows:
Accelerate (up arrow) : Press this to make the speedometer go faster. The longer you hold the key down, the faster the virtual vehicle goes.

Turn (left/right arrows) :Hold down a turn direction to blink the turn signals. Reverse Engineering the CAN Bus

Lock (left shift), Unlock (right shift): This one requires you to press two buttons at once. Hold down the left shift and press a button (A, B, X, or Y) to lock the corresponding door. Hold down the right shift and press one of the buttons to unlock a door. If you hold down left shift and then press right shift, it will unlock all the doors. If you hold down right shift and press left shift, you'll lock all the doors.

Sniffing the CAN frames generated by ICSim
Command : *cansniffer -c vcan0*

In ICSim the virtual CAN message for signals using left and right arrow key is using joystick in virtual instrument cluster. The signal ARB ID is 188 with byte value is 4.
Payload Byte Value: Right Signal 02


File Actions Edit View Help
nishant@kalinishant: ~/ICSim
nishant@kalinishant: ~/ICSim
nisha

```
53|ms | ID  | data ...     < vcan0 # l=20 h=100 t=0 slots=36 >
00000 | 039 | 00 0C                     ..
00002 | 095 | 80 00 07 F4 00 00 00 08 ........
00002 | 133 | 00 00 00 00 A7            .....
00002 | 136 | 00 02 00 00 00 00 00 2A .......*
00002 | 13A | 00 00 00 00 00 00 00 28 .......(
00002 | 13F | 00 00 00 05 00 00 00 2E ........
00000 | 143 | 6B 6B 00 FF               kk..
00002 | 158 | 00 00 00 00 00 00 00 19 ........
00002 | 161 | 00 00 05 50 01 08 00 1C ...P....
00002 | 164 | 00 00 C0 1A A8 00 00 04 ........
00002 | 166 | D0 32 00 18               .2..
00002 | 17C | 00 00 00 00 10 00 00 21 .......!
00000 | 183 | 00 00 00 09 00 00 10 3F .......?
00000 | 188 | 02 00 00 00 00 00        ......
00002 | 18E | 00 00 6B                  ..k
00002 | 191 | 01 00 10 A1 41 00 0B     ....A..
00000 | 1A4 | 00 00 00 08 00 00 00 10 ........
00000 | 1AA | 7F FF 00 00 00 00 68 10 ......h.
00000 | 1B0 | 00 0F 00 00 00 01 57     ......W
00018 | 1CF | 80 05 00 00 00 1E        ......
00004 | 1D0 | 00 00 00 00 00 00 00 0A ........
00018 | 1DC | 02 00 00 1B               ....
00019 | 21E | 03 E8 37 45 22 06 2F     ..7E"./
00000 | 244 | 00 00 00 01 13            .....
00019 | 294 | 04 0B 00 02 CF 5A 00 2C .....Z.,
00093 | 305 | 80 17                     ..
00101 | 309 | 00 00 00 00 00 00 00 A2 ........
00102 | 320 | 00 00 12                  ...
00102 | 324 | 74 65 00 00 00 00 0E 1A te......
00106 | 333 | 00 00 00 00 00 00 1E     .......
00102 | 37C | FD 00 FD 00 09 7F 00 1A ........
00029 | 405 | 00 00 04 00 00 00 00 1A ........
00033 | 40C | 03 31 33 38 34 39 00 0D .13849..
00028 | 428 | 01 04 00 00 52 1C 10     ....R..
00033 | 454 | 23 EF 09                  #..
01002 | 5A1 | 96 00 00 00 00 00 62 2F ......b/
```

Steps to Replay Attack process:
Step 1: At first in the linux terminal use to open the ICSim instrumental cluster and controller to pass the CAN message.

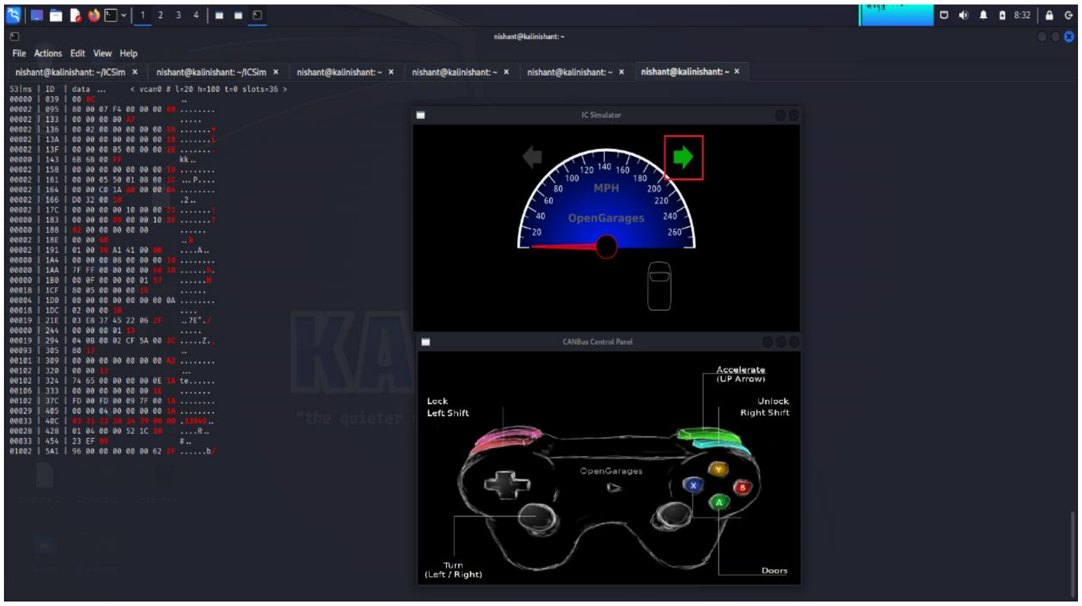
Command: *cangen vcan0 -g 10 -I 188 -L 6 -D 02 00 00 00*

Step 2: Using the CANDUMP tool to store the CAN message from ICSim process.

Command : *candump -L vcan0 > "replay_attack.log"*

Step 3: Using the CANPLAYER tool to process the replay attack in instrument of ICSim to watch the attack process.

Command : *canplayer -I "replay_attack.log"*



```
53|ms | ID  | data ...     < vcan0 # l=20 h=100 t=0 slots=36 >
00000 | 039 | 00 0C                      ..
00002 | 095 | 80 00 07 F4 00 00 00 08    ........
00002 | 133 | 00 00 00 00 A7             .....
00002 | 136 | 00 02 00 00 00 00 00 2A    .......*
00002 | 13A | 00 00 00 00 00 00 00 28    .......(
00002 | 13F | 00 00 00 05 00 00 00 2E    ........
00000 | 143 | 6B 6B 00 FF                kk..
00002 | 158 | 00 00 00 00 00 00 00 19    ........
00002 | 161 | 00 00 05 50 01 08 00 1C    ...P....
00002 | 164 | 00 00 C0 1A A8 00 00 04    ........
00002 | 166 | D0 32 00 18                .2..
00002 | 17C | 00 00 00 00 10 00 00 21    .......!
00000 | 183 | 00 00 00 09 00 00 10 3F    .......?
00000 | 188 | 02 00 00 00 00 00          ......
00002 | 18E | 00 00 68                   ..h
00002 | 191 | 01 00 10 A1 41 00 0B       ....A..
00000 | 1A4 | 00 00 00 08 00 00 00 10    ........
00000 | 1AA | 7F FF 00 00 00 00 68 10    ......h.
00000 | 1B0 | 00 0F 00 00 00 01 57       ......W
00018 | 1CF | 80 05 00 00 00 1E          ......
00004 | 1D0 | 00 00 00 00 00 00 00 0A    ........
00018 | 1DC | 02 00 00 1B                ....
00019 | 21E | 03 E8 37 45 22 06 2F       ..7E"./
00000 | 244 | 00 00 00 01 11             .....
00019 | 294 | 04 0B 00 02 CF 5A 00 2C    .....Z.,
00093 | 305 | 80 17                      ..
00101 | 309 | 00 00 00 00 00 00 00 A2    ........
00102 | 320 | 00 00 12                   ...
00102 | 324 | 74 65 00 00 00 00 0E 1A    te......
00106 | 333 | 00 00 00 00 00 00 1E       .......
00102 | 37C | FD 00 FD 00 09 7F 00 1A    ........
00029 | 405 | 00 00 04 00 00 00 00 1A    ........
00033 | 40C | 03 31 33 30 34 39 00 0D    .13049..
0002B | 42B | 01 04 00 00 52 1C 10       ....R..
00033 | 454 | 23 EF 09                   #..
01002 | 5A1 | 96 00 00 00 00 00 62 2F    ......b/
```

Reference :

- https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial.
- The Car Hacker's Handbook by Craig Smith.