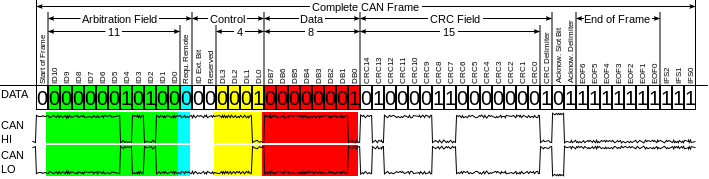
Vehicle Hacking Setup Guide

Shawn Nicolen



*Structure of a CAN Frame.*

*Source: Daniel Endres via Wikimedia Commons. This image is licensed under the* [*Creative Commons*](https://en.wikipedia.org/wiki/en:Creative_Commons) *Attribution-Share Alike 3.0 Unported license.*

# 

# 

# 

[**Introduction**](#_63tjn0sspyfx) **4**

[A Note of Caution](#_wo4o4gb1tpc5) 4

[Special Thanks](#_skyfrod8kg0t) 4

[Hardware Requirements](#_82dm8131lbk) 4

[Linux Requirements](#_81jwav47w6j4) 4

[Ubuntu](#_32ssxpik0czk) 4

[Mint](#_4p85j82q88w) 4

[Kali](#_2ojtwsk7hu2n) 4

[**Virtual CAN Interface Setup**](#_ij23v5mvbno5) **4**

[**Physical CAN Interface Setup**](#_uf2kqajgbyh3) **6**

[Common Pitfalls](#_m0648ilppak) 8

[References](#_405mbgtjh6we) 9

[**Related Applications Setup**](#_kfagr1xa6xg6) **9**

[SocketcanD](#_xajqtf2cuppv) 9

[Kayak](#_y25zih15uehd) 10

[Installing Kayak](#_4p2gpem5ig3l) 10

[From Source (Git)](#_iphed0ig1asf) 10

[From Binary](#_qwe8zwes5hc9) 11

[Running Kayak](#_sgex3zexrlyp) 11

[References](#_6lmpqxa98dxr) 11

[ICSIM](#_8wo3zroosgy9) 11

[References](#_m7sp5zewdlxq) 12

# 

# 

# Introduction

This guide is intended to guide you through the setup of a linux system to communicate with the CAN bus of a vehicle. These can be taken as step by step instructions for someone with little to no experience with vehicle technologies and only a little experience using Linux.

## A Note of Caution

A modern vehicle is a computer attached to a complex and mobile physical system. Misuse of vehicle cybersecurity techniques may create situations which are dangerous to yourself or others. Exercise extreme caution with the techniques described here, and never attempt the steps described while on public roads.

## Special Thanks

Special thank you to Craig Smith ([@OpenGarages](https://twitter.com/opengarages)), the author of the Car Hackers Handbook, for which this guide was intended to serve as supporting material. For more information about this excellent resource visit <http://opengarages.org/handbook/>.

## Hardware Requirements

The examples below use the excellent CANTact created by Eric Evenchick ([@ericevenchick](https://twitter.com/ericevenchick)), an open source CAN to USB interface. For more information, visit <http://linklayer.github.io/cantact/>.

## Linux Requirements

All of the examples below assume you have a fresh install of Linux and a readily available direct internet connection.

### Ubuntu

The examples given in this document were created with a fresh install of Ubuntu 14.04.4.

### Mint

Since Mint is based on Ubuntu the examples given below should work without issue.

### Kali

It is possible to use the steps below on Kali Linux but you will need to update your apt-get repositories to be able to use the packages mentioned below. A tool called Katoolin can be used to do so and is available at: <https://github.com/LionSec/katoolin>.

# Virtual CAN Interface Setup

1. Install can-utils

sudo apt-get update

sudo apt-get install can-utils

2. Load Kernel Modules

You will need to ensure your kernel has the CAN modules enabled. This is done with modprobe:

For physical CAN devices:

sudo modprobe can

For virtual CAN devices:

sudo modprobe vcan

3. Add the virtual CAN device and enable it with the following commands:

sudo ip link add dev vcan0 type vcan

sudo ip link set up vcan0

4. At this point the virtual CAN device should be up and running. Let’s test it…

Open two terminal windows. In terminal window 1 type:

candump vcan0

In terminal window 2 type:

cansend vcan0 123#DEADBEEF

If everything is set up correctly you should see the CAN frame from terminal window 2 arrive in terminal window 1.

To generate continuous stream of random CAN traffic, type the following in terminal 2:

cangen vcan0

You should see a stream of random CAN data arrive in terminal 1.

**Note:** I don’t recommend running this command against a real vehicle during later exercises, as there is no telling what the random traffic this command generates will do to the target vehicle.

The *Car Hackers Handbook* has an excellent summary of can-utils commands on page 41. Give a few of them a try. Observe the traffic you send and receive. Try creating a log file of some sample CAN commands. Familiarize yourself with can-utils. It is much safer and easier to do this with a virtual device than an actual physical vehicle.

# Physical CAN Interface Setup

Setup of a physical device using the CANtact is a little different. In this case we are not using a virtual device but rather a serial hardware device plugged into our computer via USB.

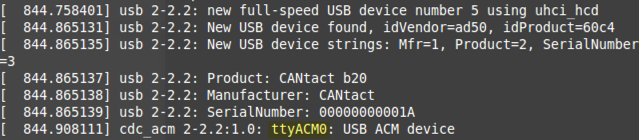
1. You will need to ensure your kernel has the CAN modules enabled. This is done with modprobe:

For physical CAN devices:

sudo modprobe can

1. Make sure the physical PIN out configuration is set correctly on the CANtact. The most common configuration is 3-5-1.
2. Plug the CANtact device into the computer’s USB port.
3. Determine the device name that your CANtact has registered itself as in /dev.

dmesg



*Sample output of dmesg after plugging in the CANtact. The name assigned to this device is highlighted.*

1. Observing the output of the dmesg command I found that my CANtact device was registered as ttyACM0.

You could also follow the results to syslog to see the name of the device.

tail -f /var/log/syslog

1. Next, we will create a serial can device link to this device hardware. This can be done as a daemon or directly via interactive session. In Linux a daemon is, “a type of program on Unix-like operating systems that runs unobtrusively in the background, rather than under the direct control of a user, waiting to be activated by the occurrence of a specific event or condition.” (<http://www.linfo.org/daemon.html>)

In these examples *<CANtact device>* would be replaced with my device name, ttyACM0, the name of the device as we saw in /dev.   
  
The slcand command connects the serial CAN device as a daemon while slcan\_attach will attach the device directly. The specifics of the command below are detailed in the *Car Hackers Handbook*, page 39.

sudo slcand -o -s6 -t hw -S 3000000 /dev/*<CANtact device>*

Or you can just attach the device directly without using the daemon:

sudo slcan\_attach -o -s6 -w /dev/ttyACM0

Either of these will create a linked serial device named “slcan0” which we will reference in later sections to interface with a physical CAN connection. Much as we set up a virtual CAN device earlier (named vcan0) here we have created an interface to a physical CAN (named slcan0).

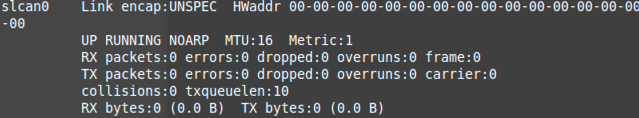
1. Now that we have created a serial link to this new CAN device we need to bring up its network interface.

sudo ip link set up slcan0

We can check to see that this is set up correctly using the ifconfig command.

sudo ifconfig slcan0

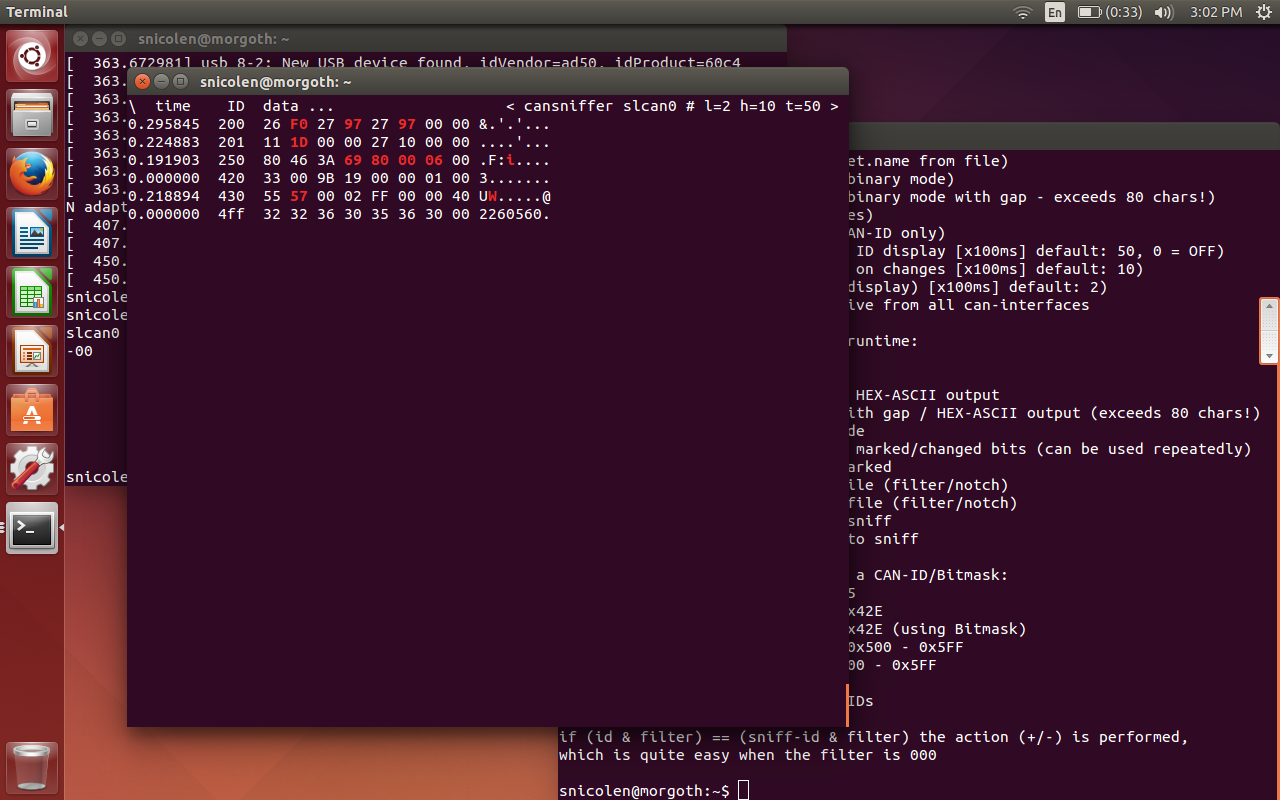
If this returns anything other than an error your device is likely set up correctly.



1. At this point we are ready to plug into the OBDII interface on a target vehicle.

**Note:** For this exercise we will NOT be writing data to the vehicle CAN bus – only reading data. The *Car Hackers Handbook* has an excellent summary of can-utils commands on page 41.

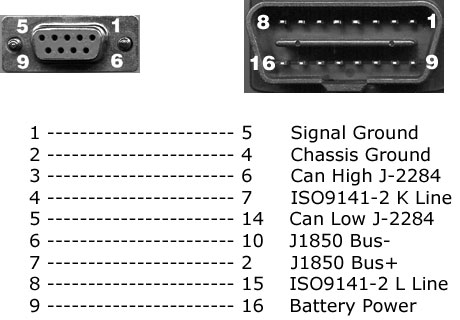
* 1. candump
     1. Dumps the raw packets read from the target bus. It can be used to create log files of CAN traffic for later analysis.
  2. cansniffer
     1. An interactive traffic viewer which will be our main means of watching live CAN traffic.



*Sample output of cansniffer. It works! Recently changed values are highlighted in cansniffer using the “c” option.*

## Common Pitfalls

1. Make sure that the pin-out is set correctly on your CANtact. For most American made vehicles the 3-5-1 pin configuration will match the physical pinout configuration of the target vehicle. Other pin configurations may be needed for European or Japanese manufactured vehicles.



*Source:* [*https://www.obd2allinone.com/products/obd2cable.asp*](https://www.obd2allinone.com/products/obd2cable.asp)

1. Make sure that the correct CAN bus baud rate and speed are set. For CAN-H connections you will want to the –s6 option, which corresponds to 500kbps. See page 39 of the *Car Hackers Handbook* for more information.
2. If you unplug and re-plug the CANtact into the usb port you may need to check /dev for a new device name (usually incremented by 1) and walk through the CANtact set up process all over again using the new device name.
3. When in doubt unplug everything and reboot.

## References

1. <http://elinux.org/Bringing_CAN_interface_up>
2. <https://discuss.cantact.io/t/has-anyone-gotten-their-cantact-to-work/37>
3. <https://discuss.cantact.io/t/using-can-utils/24>
4. <https://discuss.cantact.io/t/quick-start-guide/18>
5. <http://pinoutsguide.com/CarElectronics/car_obd2_pinout.shtml>

# Related Applications Setup

## SocketcanD

Socketcand is a linux daemon that provides access to CAN interfaces on a machine via a network interface. It is required by other applications listed later in this guide.

In order to compile SocketcanD you may need to install some development packages:

sudo apt-get install libconfig-dev

sudo apt-get install libreadline6-dev

sudo apt-get install libssl-dev

sudo apt-get install autoconf

And, in order to download SocketcanD you will need to install git:

sudo apt-get install git

Obtain the source code for SocketcanD using git:

git clone http://github.com/dschanoeh/socketcand.git

Now, compile it...

cd socketcand

autoconf

./configure

make clean

make

sudo make install

Lets try attaching socketcand to a can interface. For this, let's use a virtual can interface from one of our earlier examples: vcan0. Remember to set up your virtual CAN interface before trying this next command.

socketcand -v -i vcan0

### References

1. https://github.com/dschanoeh/socketcand

## Kayak

Kayak is a free application for CAN bus diagnosis and monitoring by Jan-Niklas Meier. Download Kayak from its home page. You can either use git to obtain the files and compile it yourself or download the binary. For either, see:

* <http://kayak.2codeornot2code.org/>

### Installing Kayak

#### From Source (Git)

Run the following to obtain Kayak using git:

git clone git://github.com/dschanoeh/Kayak

Then to build it you will need Apache Maven, a build manager for java projects:

sudo apt-get install maven

Finally, you can built the project:

cd Kayak

mvn clean package

To run Kayak you can find it in:

Kayak/application/target/kayal/bin/kayak

#### From Binary

Go to the Kayak website at:

<http://kayak.2codeornot2code.org/>

Download the binary installer, and run it.

### Running Kayak

Before running Kayak make sure that SocketCanD is running and attached to a live CAN interface, either virtual or real. In the examples below we will use a virtual interface for testing purpose. If you are following along make sure you have created the virtual CAN interface per earlier instruction.

socketcand -i vcan0

### References

1. Kayak Tutorial: <http://kayak.2codeornot2code.org/tutorial.html>
2. <http://kayak.2codeornot2code.org/>

## ICSIM

For this exercise you can use the keyboard to generate simulated CAN traffic. This tool also supports the use of a joystick to do this as well.

sudo apt-get install joystick

Jstest-gtk is a program that can be used to calibrate the joystick

sudo apt-get install jstest-gtk

ICSim will require a few libraries to run:

sudo apt-get install libsdl2-dev

sudo apt-get install libsdl2-image-dev

sudo apt-get install can-utils

Download ICSim and compile it:

You can run the following commands to setup a virtual can interface:

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

You can now use icsim to simulate can traffic to the virtual interface:

icsim vcan0

### References

1. ICSIM: <https://github.com/zombieCraig/ICSim>