# Setting up Linux

* If you do not have the can-utils package, create a directory in your user/Downloads folder in Linux by issuing the command: *mkdir can-utils*
* Change to the directory: *cd can-utils*
* Clone the can-utils git repository: *git clone* [*https://github.com/linux-can/can-utils.git*](https://github.com/linux-can/can-utils.git)
  + Git will clone the files to your local directory
* Compile the packages by typing:
  + *make*
  + *sudo make install*
* The can-util binaries will now be compiled, including:
  + Cansend
  + Candump
  + can

# Infotainment Lab 1 (Easy)

CAN Bus is a ubiquitous communication bus that is used to transmit information between the various Electronic Control Units (ECUs) in a vehicle. CAN bus is designed to be a cheap, reliable and inexpensive communication network, but was designed with no inherent security mechanisms. Verifying the source of a message on the CAN bus is difficult, and not generally done on a vehicle platform. This allows any attacker with physical access to the CAN bus to impersonate any device, causing other devices to respond as if the messages being transmitted by the attacker are legitimate. We will explore this capability by causing our infotainment unit to display a message.

## Setup

This lab uses a CAN device called a CANable, coupled with Linux’s Socketcan tools. Socketcan allows CAN devices to be used like any other network device on a Linux system. You will need the can-utils software package installed on your Linux system. Once installed, connect the CANable device to the infotainment system wiring harness as pictured, with the yellow wire connected to CANL and the green wire connected to CANH. Connect the micro USB cable from the CANable to your computer.

A green circuit board with red and black components

Description automatically generated

Turn the infotainment system on by connecting the power supply to the barrel connector on the wiring harness and turning the switch on. There are two types of power switches depending on your particular infotainment unit.

Switch Type 1



Switch Type 2

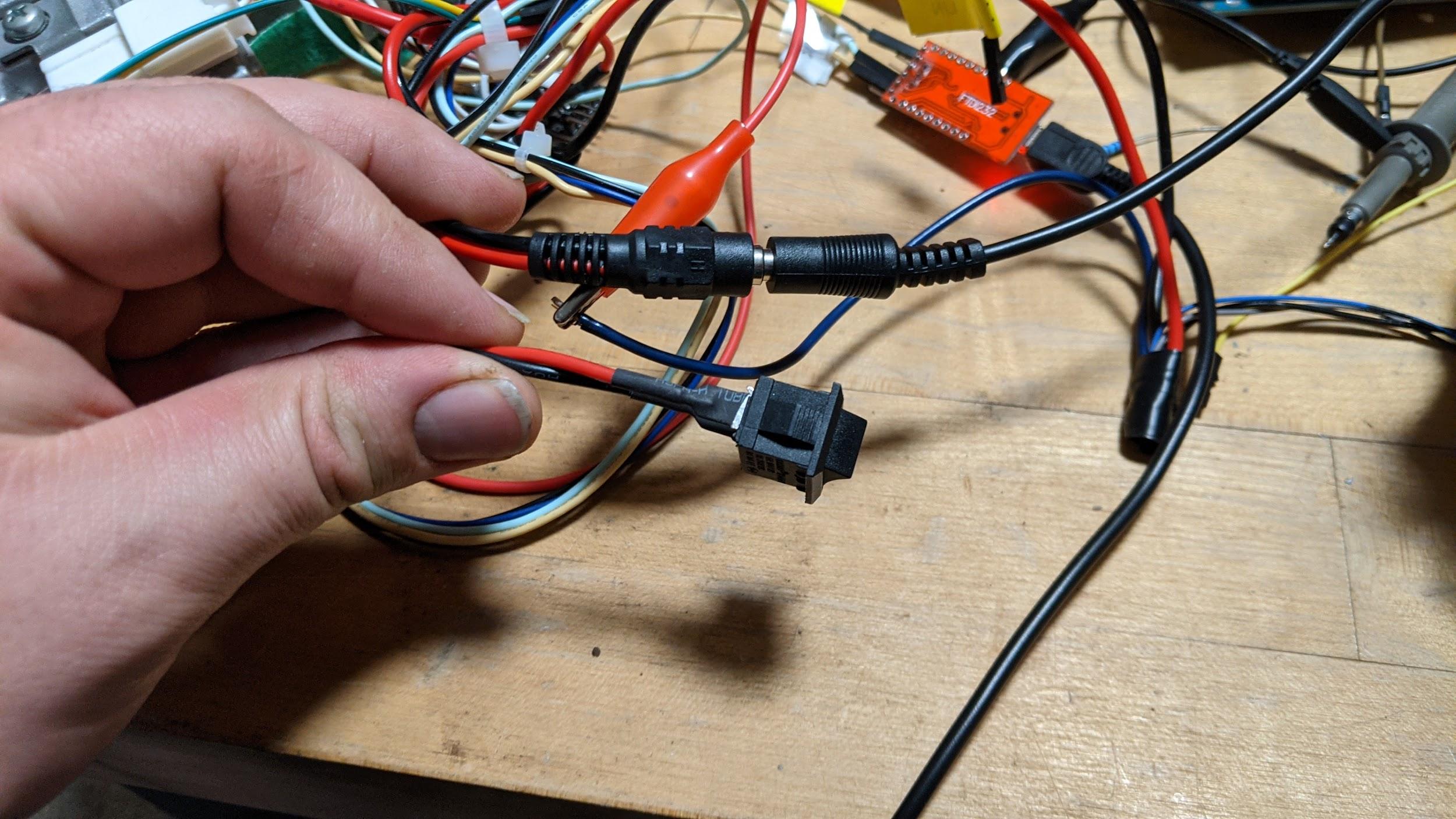


Photo of wire connections leading from the wiring harness to the back of the infotainment system.

A close up of a device

Description automatically generated

Set up the CAN network on the Linux machine by running these commands in a terminal window.

sudo ip link set can0 type can bitrate 500000

sudo ip link set up can0

## Instructions

Linux can-utils has several utilities for reading messages from and sending messages on the CAN bus. Start by looking at the CAN traffic being sent by the infotainment system by running the following command:

candump can0

You should see CAN messages being printed to the screen. Press CTRL-C to quit the candump program.

Now, send a specific message on the CAN bus by running the following command:

cansend can0 445#872c4763

Watch the infotainment screen. This CAN message encodes the backup warning message including proximity to objects. The CAN message is identified by an arbitration ID, which for this message is the value 0x445. The data of this message is the hex string 872c4763. Try changing this data to see how the proximity warning display changes (for example, try sending the data section as 872c4993 to see if the display changes). The arbitration ID needs to be left as 445.

## Conclusion

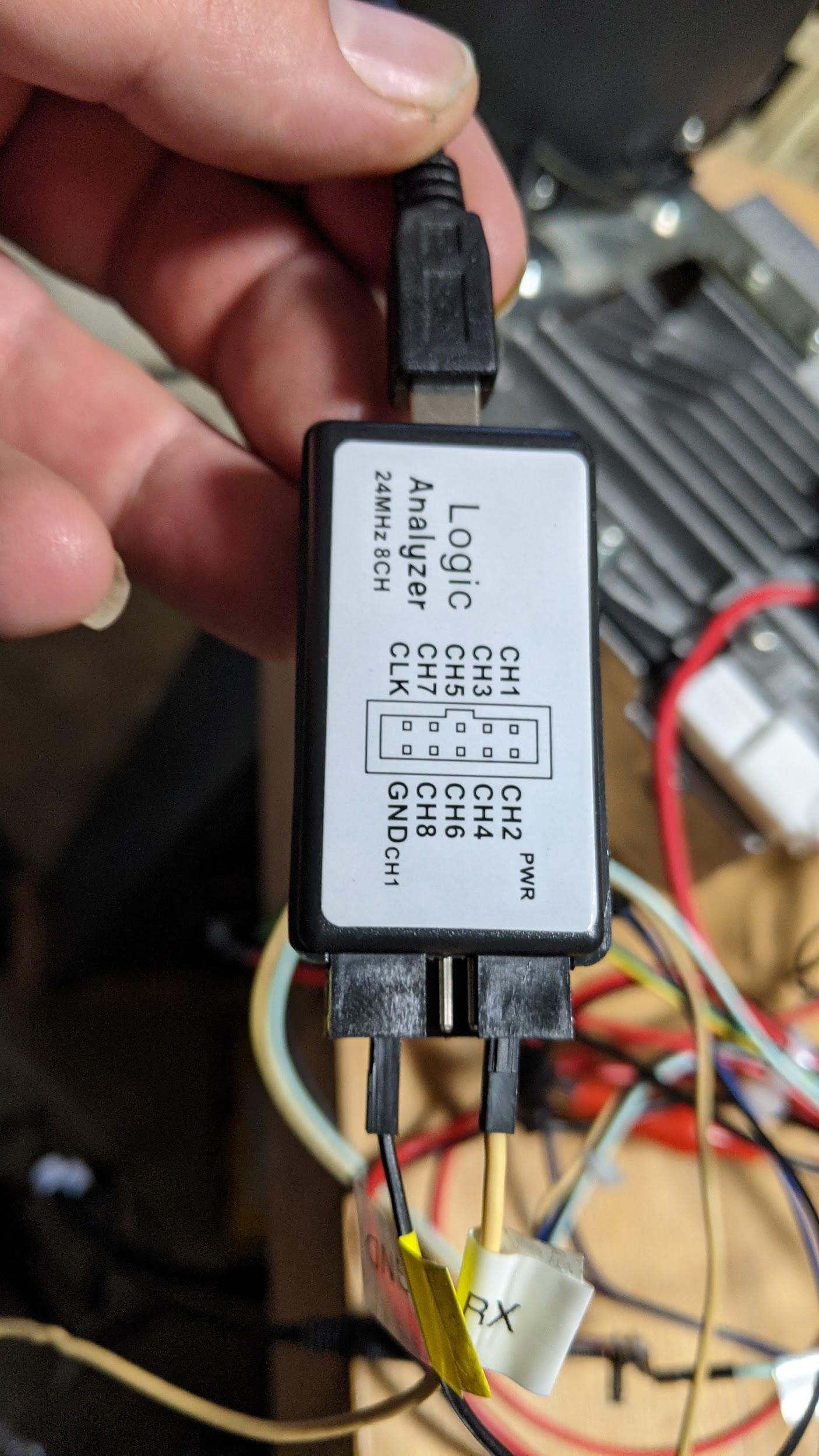
Since CAN provides no protection against replay attacks, nothing prevents an attacker with physical access from sending arbitrary CAN messages to cause ECUs to misbehave. This misbehavior could include displaying spurious warning messages, and even control of steering, braking and acceleration in some vehicles.

# Infotainment Lab 2 (Medium)

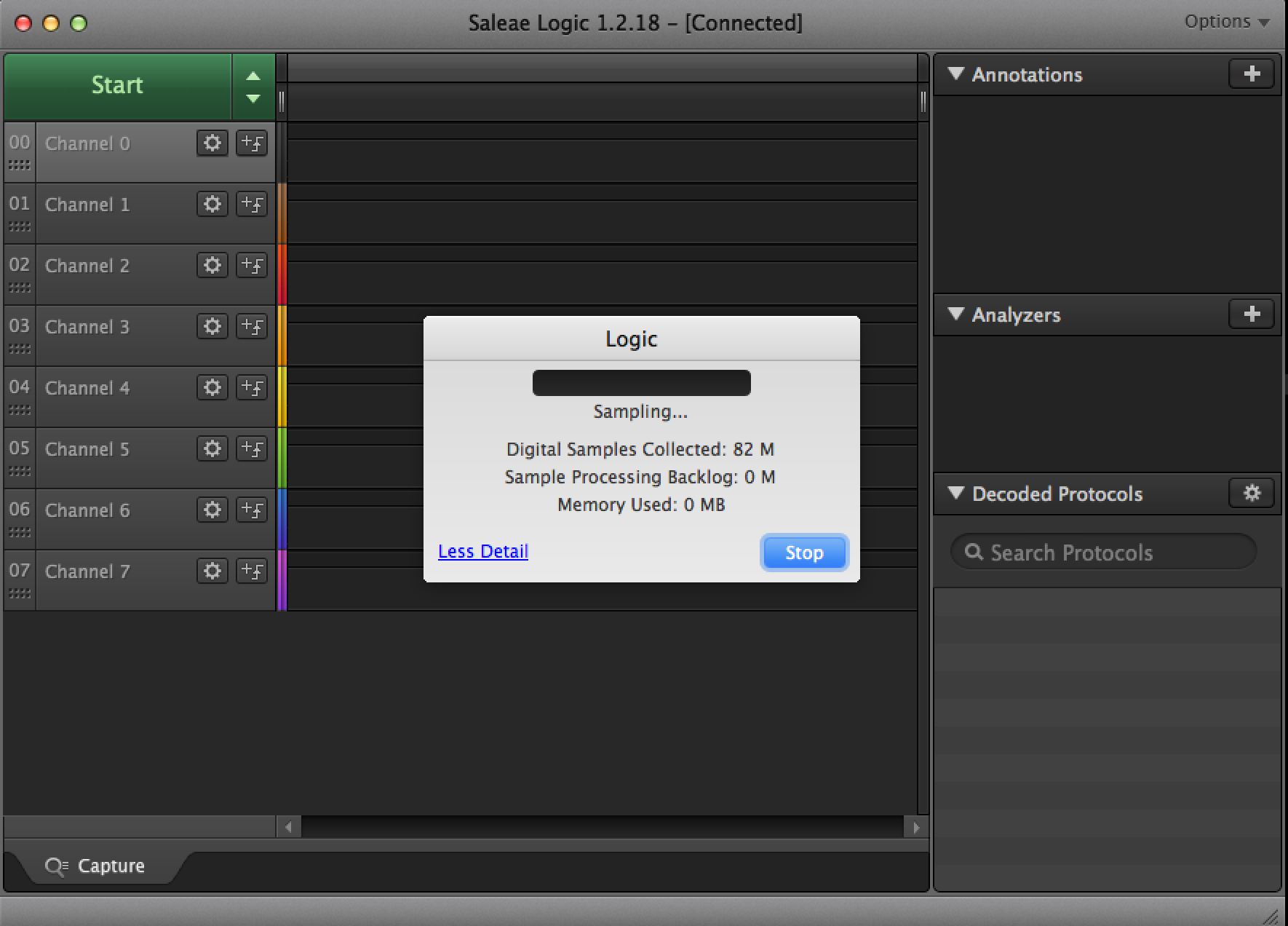
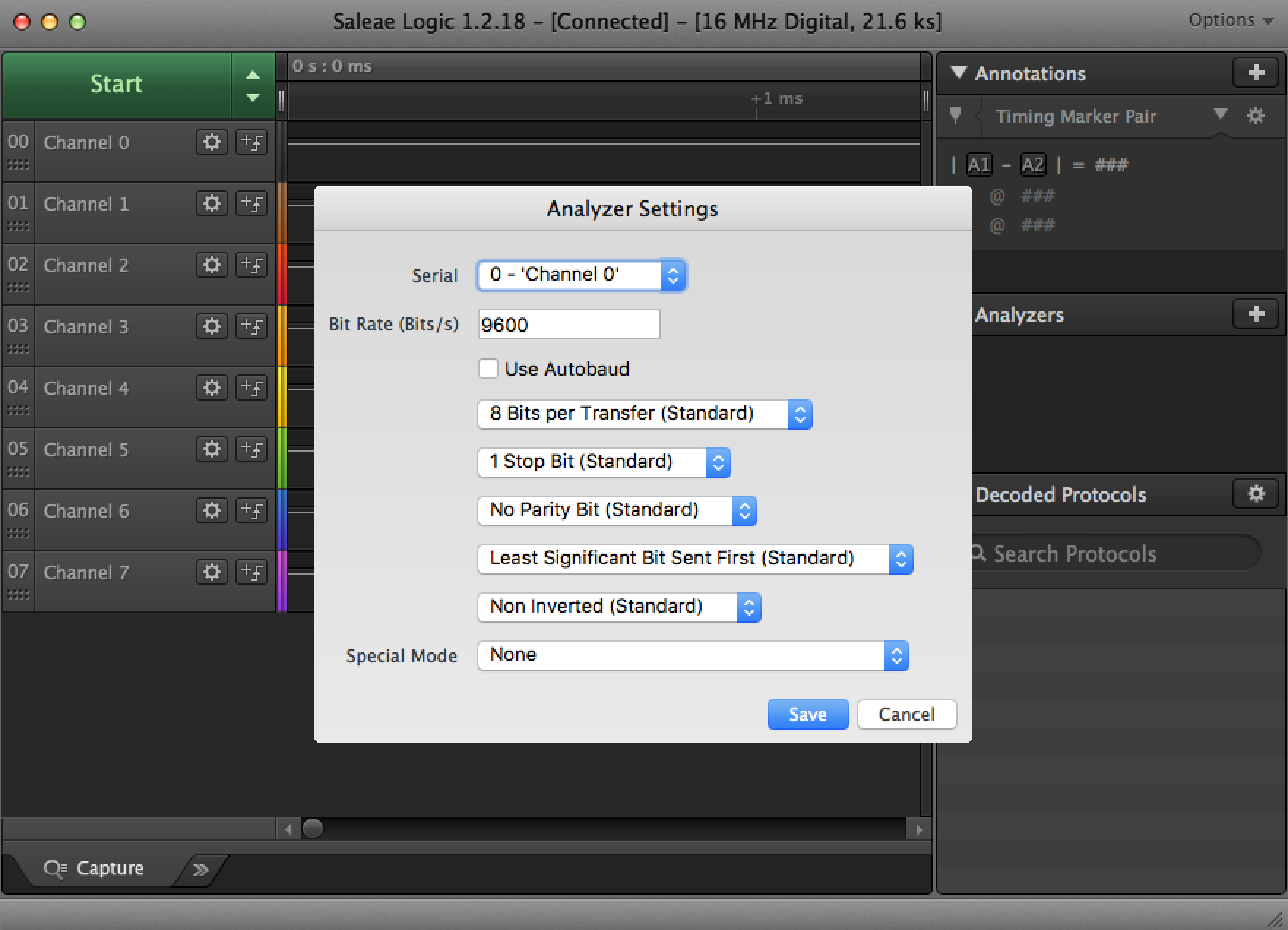
During product development, developers need debugging access to the system. This debugging access often happens over a serial console. Frequently, this serial debugging access isn’t disabled, or isn’t fully disabled once the system goes into production. Using a logic analyzer to find these unused serial ports is often a productive early step when hacking hardware.

## Setup

Download and install the Saleae LOGIC software from <https://www.saleae.com/downloads/>. Connect the line labeled RX to pin 1 of the logic analyzer and connect the line labeled GND to the GND pin. Use the mini USB cable to attach the logic analyzer to the computer as pictured. Plug the power supply into the infotainment system, but do not turn the infotainment system on.



## Instructions

1. Turn the infotainment unit off if it is on.
2. Start collecting data from the Logic Analyzer by pressing the “Start” button at the top left corner. You can change sampling parameters by pressing the down arrow on the right side of the start button. Since serial consoles are slow, try sampling for 30 seconds at a rate of 1 Mbps.
3. Turn the power on to the CMU. Wait for data collection to finish:
4. If you zoom out, you should see pulses on Channel 0 (Pin 1 on the logic analyzer corresponds to Channel 0 in the software). Press the + arrow in the Analyzers section on the right and add the following “Async Serial” analyzer with a bit rate of 115200. You can also experiment with using the Autobaud setting to try to automatically guess the bit rate:
5. In the bottom right corner of the screen you should see the decoded serial signal. It should look like boot messages from a linux system.

## Conclusion

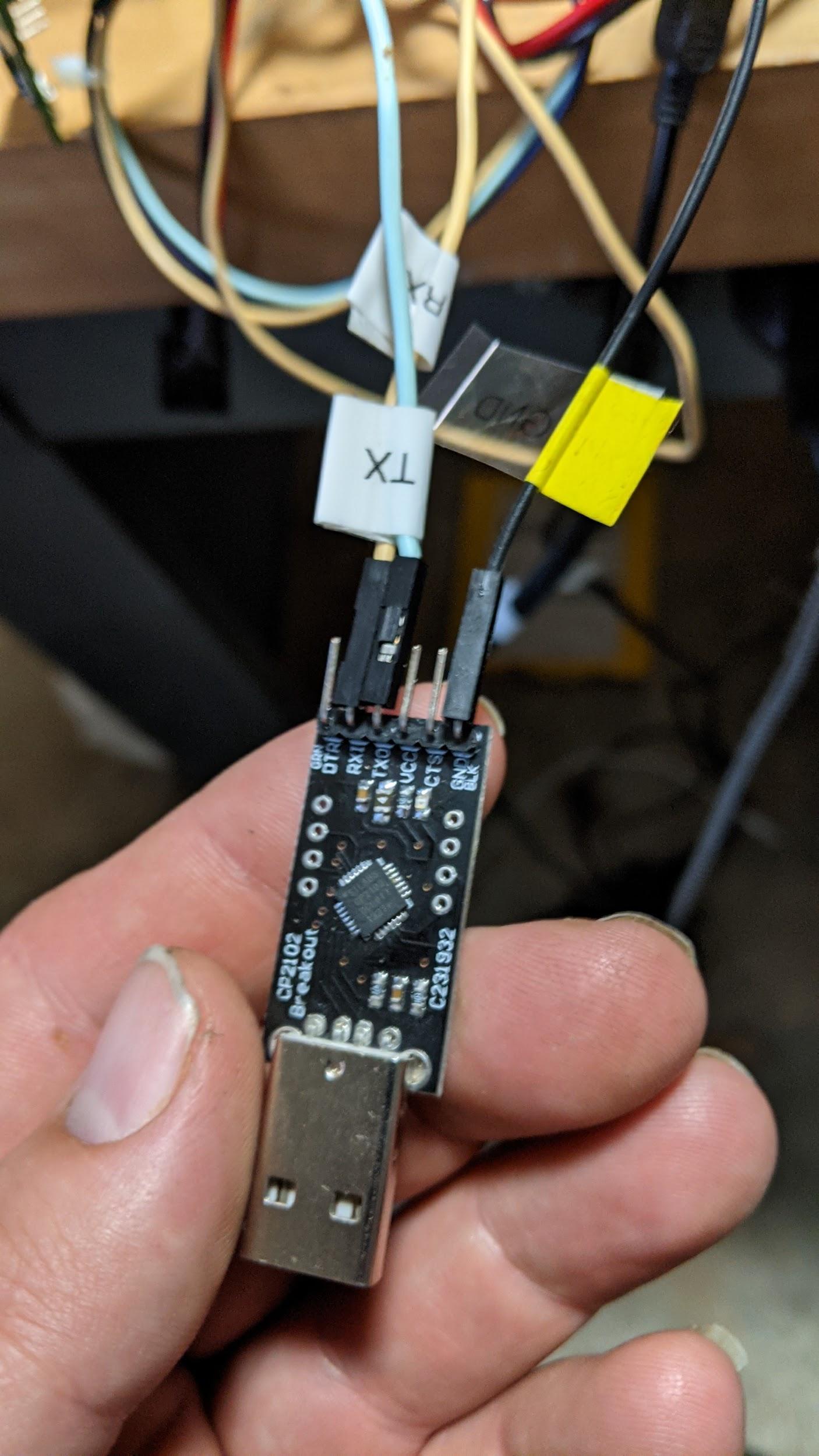
Logic analyzers are useful tools for investigating unknown signals and determining what they are. We can use the knowledge gained in this lab about how to access the serial console to allow us to complete the next lab.

# Infotainment Lab 3 (Hard)

Now that we have found the debug serial console, let’s explore it and see what access it gives us.

## Setup

Connect the USB-Serial adapter to the infotainment system as pictured (RX to RX, TX to TX and GND to GND).



Plug the USB-Serial adapter into the computer. Turn the infotainment system on.

## Instructions

1. Type the following command on the linux console to connect to the serial console:  
   sudo screen /dev/ttyUSB0 115200
2. We will see a lot of debugging messages being printed to the screen. For the first minute or two that the system is powered on there are a lot of messages being printed. After the first couple of minutes the messages slow, but do not stop. It’s important to note that these debugging messages do not affect any commands we are about to run on the infotainment system, so continue typing the following commands even if debug messages obscure or seem to overwrite what you are typing.
3. Press enter to get to the login screen. Log in with the following credentials (Note: the username is root if your infotainment system has a label on it that says “Username: root”):  
   Username: cmu  
   Password: jci
4. Sometimes debugging scripts are left on systems. If we look in the directory /jci we will see a great number of these scripts on this system. One that stands out on further investigation is the jci-wifiap.sh script which appears to set up a wireless access point. This infotainment unit isn’t supposed to have Wi-Fi, but the Bluetooth chip used is a combination Wi-Fi/Bluetooth chip and it appears the developers used the Wi-Fi access point functionality for debugging access. Let’s start the access point and see if we can connect to it. Run the following command on the infotainment system:  
   /jci/bin/jci-wifiap.sh start
5. Connect to the new access point set up by the infotainment system. The name of this access point is CMU-MACADDR, where MACADDR is the MAC address of the WiFi chip on the unit. The MAC address is printed on a label on the bottom of the infotainment unit.
6. Once connected, try to SSH into the infotainment system by running the following command (again, if your infotainment unit has a label that says Username: root substitute root for cmu in the following command):  
   ssh cmu@192.168.53.1  
   Enter jci for the password. The SSH console is easier to use than the serial console since you don’t have debugging messages being printed to the screen, though the responsiveness of the SSH console is not very good and will often seem to hang.

## Conclusion

Finding debugging scripts and programs on embedded systems can be a powerful way to understand and get access to a system.