Arsenal Labs 2024
RF Hacking on the Road: Logging Tire Sensors

Hello! In this short project, you'll see how to build an RF logger to automatically extract payload data from tire pressure monitor transmissions. These sensors are in nearly every tire on the road, periodically transmitting the tire pressure and (sometimes) temperature to the vehicle for display and alert purposes.

Although this project is somewhat basic, it does show you the process of transforming radio signals into data bits and sending them to a Python script for processing. These are both powerful capabilities of GNU Radio that can often be tricky to learn.

You won't have to grapple with a blank slate, each step will include a starter project to which you merely need to make adjustments. If you get stuck at any point, raise your hand for a helpful nudge. You can also turn the page to see the solution for the current step.

Let's get started!

Step 1 - Capture a TX from a single TPM

Before you can start reverse engineering a radio signal, you have to find it. By "find" we're not talking about a physical location, but rather a radio frequency. At any given moment, the place you're currently sitting had dozens of radios signals going through it: WiFi, cellular voice/data, Bluetooth, broadcast FM radio, and the list goes on.

The reason all these radio signals can coexist is because they're occupying different ***frequencies***. When we try to "find" the target signal, then, we're really talking about the frequency it's using.

Fortunately, some devices have a lot of information publicly available. TPMS devices are especially well publicized. A quick search of easily accessible resources will reveal that TPMS transmissions occur at 315 MHz in the US, and can use one of two modulation types: On Off Keying (OOK) and Frequency Shift Keying (FSK).

Before building a logger, we need to start by examining a signal from a real TPMS and then reversing it. It's easier to inspect a captured signal on disk than to try to look at a live transmission, so let's start by capturing a transmission.

The facilitator of this lab has a special device used by auto mechanics to program and debug these sensors. One of the features of this tool is the ability to force a sensor to transmit. He'll do so periodically for lab participants to capture a signal. To capture:

1) Type the following into your terminal window but don't hit enter yet:
   `uhd_rx_cfile -A RX2 -f 315e6 -N 5000000 user_315M_1Msps.cf32`
2) Flag down a presenter and ask them to activate the diagnostic device, forcing a TPMS transmission
3) As soon as they give you the signal, press Enter
   (The "-N 5000000" will capture 5 seconds (5 million samples at 1 Msps) of spectrum)

Solution 1

If you have issues capturing a file, you can copy the pre-captured file from the solutions folder into your working directory:

```
cp solution/quickstart_315M_1Msps.cf32 ./user_315M_1Msps.cf32
```

Step 2 - Reverse the Modulation and Timing

The next step is to determine what kind of modulation scheme is used by the sensor. We'll use a pre-made GNU Radio Companion flowgraph to figure this out. Type the following to open it:

```
gnuradio-companion 02_demod.grc
```

This flowgraph performs both OOK and FSK demodulation. The results of both are displayed in a GUI so you can see which produces a reasonable digital signal.

The GUI contains two sliders at the top: threshold and squelch_level. The threshold control relates to the OOK demodulation. Gradually increase this until you see a stable waveform appear.

Next, gradually increase the squelch_level to improve the FSK demodulation. As you increase it, the noise should diminish and a stable waveform should appear.

Choose the modulation that best appears to produce a digital waveform. Then measure the timing on the waveform by clicking and dragging to zoom into one of the shortest transitions. Measure the timing in microseconds either by referring to the horizontal axis markings or by hovering the mouse over both sides of the short transition and calculating the time difference.

Solution 2

The FSK produces the better demodulation. Once the squelch_level exceeds roughly -50 (the exact level will depend on your local noise conditions) you should see a clear and stable digital waveform with minimum transition times of 100 microseconds.

After increasing the threshold slightly, the OOK display will only show a single, long pulse. This is not a digital waveform but rather the transmission envelope of the FSK signal.

Step 3 - Extract the Payload Data

Next, you'll use another GNU Radio Companion flowgraph to perform clock recovery, the process of extracting data bits from the demodulated waveform. Click File->Open and select `03_extract.grc`

Look closely at the notes in the flowgraph, as they describe the values you must provide for the flowgraph to function:

- symbol_time (the minimum timing measured in the last project)
- squelch_level (the optimal value from the previous project)
- select_mod (chooses OOK or FSK demodulation; base this on your choice in the last project)

When you select reasonable values for these settings, you should see a clear demodulated waveform again, as well as 1s and 0s in the lower left portion of GNU Radio Companion.

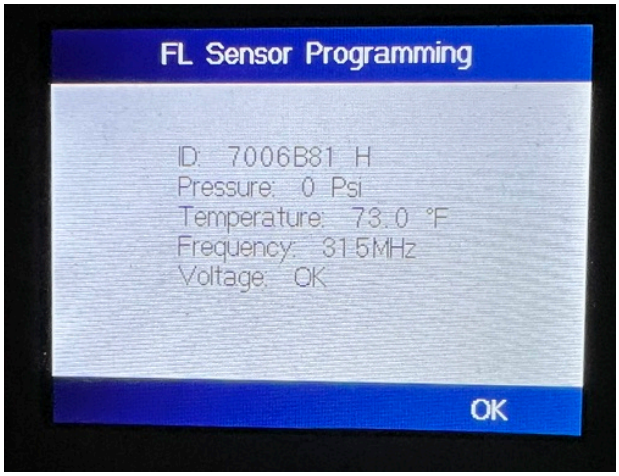
Next, we'll want to work on logging and decoding this data.

Solution 3

Set symbol_time to 100 microseconds, as measured in Step 2. Also set the squelch level to roughly -45 (you may need to adjust this slightly for your particularly noise levels). Finally, make sure to select FSK from the modulation pulldown.

Step 4 - Log the Captured Data and Find the TPM ID

Before trying out the logger on live transmissions, we'll test it on your captured-file data. We'll also examine the data to find the ID burned into the captured TPMS (see the TPMS tool display at right).



Open yet another flowgraph called `04_logger_file.grc`

This flowgraph is similar to the previous one, but an additional block at the end of the signal chain: a `ZMQ PUSH Message Sink`

This block will take the payloads extracted from the flowgraph in the previous step and send them through a socket to a Python script for further processing. The Python script we've configured is called `rf_logger.py`

Open a second terminal window, if you don't already have one open, and run the Python script with:

```
~/tpms_lab/rf_logger.py -h
```

This will show you the help for running this script. The only command line flag is for a decoding scheme. You will have to decide which of the four encoding schemes produces the best data. First, run the flowgraph. Next, switch to the second terminal window and run the logger command with one of the decode flags. If you don't have an educated guess, run with all four schemes to see which produces the most interesting data. You can type Ctl-C to quit the script at any time.

Before quitting the script for the last time, it's a good idea to kill the GNU Radio Companion execution window first, then hitting Ctrl-C in the terminal window running the script. Doing so in the opposite order will cause the GNU Radio window to hang.

Solution 4

The waveform is encoded with a Differential Manchester scheme, necessitating the “-d 3” command line option.

Step 5 - Running the Logger

Now you’re ready to log live transmission! Open the last of the flowgraphs, called `05_logger_live.grc`

This is similar to the previous flowgraph except that it uses live SDR input from the USRP rather than data from your captured file. Midway through the lab, the presenter will begin transmitting TPM signals captured from the side of a local road. The goal is to run the scanner and log any sensors using the protocol you just reversed. There will also be other TPMS signal in the mix, but your logger is not yet equipped to capture them.

The sensor you reversed above will be in the rebroadcast, as will another sensor using the same protocol. Your goal is to extract the sensor ID from the second TPMS.

Run the new flowgraph and the Python script as you did in the last step, waiting until you see the transmission in question.

Solution 5

Tips:

- make sure to run the Python script with the “-d 3” options for Differential Manchester encoding
- Extended your antenna to roughly 25 cm (10 inches), which is about 1/4 of the wavelength of the 315 MHz signal
- Adjust your squelch level down until you are just barely getting noise on your FSK demodulator output

Conclusion

We hope you had an informative and entertaining time with us! Please let us know if you have any questions or thoughts about this project or any other radio-hacking topics.

Thanks for joining us!