Basic Radio Hacking - Self Start

Hello! In this short project, you'll get a ground-level view on what it means to hack a radio system. Like many InfoSec professionals, you may already have a wealth of experience reverse engineering protocols in a non-radio context. Often you can leverage those skills for radio hacking if you can just get the radio signals transformed into bits.

Although this project is very basic, it does show you the process of getting radio signals into bits. It also shows you how to operate transmitters that send your chosen bits to take over control of the original target - or other targets.

You won't have to grapple with a blank slate, each step will have 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 or just turn to the next page to see how we solved it. You can also open the solution to each step in each project's "solution" subdirectory.

Let's get started!

Step 1 - Find the Signal

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.

Your computer should already have an instance of GNU Radio running, with some blocks and connections already on the screen. If not, please raise your hand and call someone over to get your laptop reset to Step 1 (it only takes a second).

What you see on the screen is called a ***flowgraph***, which you can run by clicking on the little play button on the middle of the toolbar. You should then see a colorful window pop up with a slider control near the top that allows you to change the frequency of the display. Try changing the frequency until you can see activity on the display that corresponds to when you (or someone else) presses the button on the garage door remote.

Then note the frequency at which you see the activity (it will be in mega-hertz, or MHz). You can then stop you flowgraph by clicking in the top-right of your pop window.

Congratulations! You're done with Step 1!

Solution 1

There are a lot of signals flying around all over the RF spectrum, but if you slide the control over to 390 MHz and press the remote button, you'll see a vertical, dotted line running down the middle of the display. That's the signal.

Step 2 - Capture the Signal (OPTIONAL)

NOTE: This is the least interesting step to perform, and we've already got a captured file for you on disk, so if you want to skip this part, that's totally OK.

Next, you want to create a copy of the radio signal on your hard drive. This will allow you to analyze the signal by processing the stored copy, instead of having to work on a real-time signal that's only available when someone is pressing that button.

There's another flowgraph on your laptop you can use to capture the signal from the garage door remote. You'll just need to tweak it a bit to work.

First, close the Step 1 flowgraph by hitting Ctrl-W. Then open the new flowgraph by clicking File->Open and clicking the 02_capture directory, after which you'll double-click the capture.grc file.

This flowgraph has not yet been configured to capture the frequency at which the garage door operates. To do so, you'll make two changes: to the flowgraph's center_freq value, and the name of the file into which you'll save your data.

Find the Variable block with its Id set to ***center_freq*** and double-click this block to open its properties. The Value is currently set to 100e6, which is exponential notation for a 100 with six zeros on the end, AKA 100,000,000. With units, this is 100 MHz. Change this to the value you found in Step 1 and click OK.

Next, find the File Sink block and double-click it. It's always a good practice to have your file names contain info like the center frequency. This part of the file name is currently x'd out, so replace the 'x' characters with the correct center frequency value and then click OK. This is actually an optional step, leaving the file name unchanged won't affect the flowgraph - it will still work. Even so, it is a good practice to use the most descriptive name possible for your radio captures.

Now get ready to press the button on the garage door remote (or ask someone else to press it). When you're ready, click the little play button to execute the flowgraph. When the button is pressed, you should see a spike in the middle of the display. Once you've seen this spike, you can stop the flowgraph. The entire time the flowgraph was running it was capturing the RF data around your chosen center frequency. You can think about it like an audio recorder, except for radio waves.

Solution 2

You should change the value of the center_freq Variable to 390e6, and the File Sink's filename to remote_01_c390M_s1M.iq

Step 3 - Tune and Demodulate

Now that you have a copy of the remote's signal sitting on your disk, it's time to see what kind of waveform it contains. For that, you'll need to perform tuning and demodulation. Again, we've got a flowgraph already built for this - you'll just need to make a few adjustments.

Close your previous flowgraph (remember Ctrl-W?) and open 03_tune_demod/tune_demod.grc. This is a radio receiver flowgraph - there's a lot going on here, but don't worry, you don't have to understand all the pieces. The goal of this step is to figure out three things: the signal's frequency, the signal's bandwidth and a threshold value, which is related to how big the signal is.

This flowgraph is already set up to use a file we captured for you, but if you want to use your own captured signal, double-click on the File Source block and select the filename you captured. Now run the flowgraph. You'll see three slider controls along the top. When you've adjusted these to reasonable values, a clean waveform will emerge. There are two plots to help you out through this process: the topmost plot is labeled "Demodulated Signals" and the bottom "Raw Capture RF."

You'll start with the frequency control. Slide it to the frequency at which you found the signal in Step 1. You can also look at the lower-most plot (Raw Capture RF) to see which frequency to use - wherever the spike appears, that's your frequency. After you set the frequency correctly, you will start to see a very rough waveform appear in the upper-most plot (Demodulated Signals).

Next, take a look at the width of the spike in the bottom plot (Raw Capture RF). Estimate roughly how wide it is by hovering the mouse over each side. Clicking on the Max Hold box may also help. What you're doing here is approximating the ***bandwidth*** of the signal. Set the channel width slider to something a bit higher than the observed bandwidth. Now, the demodulated signal in the uppermost plot should look a little sharper, a little more like a blocky, digital waveform.

The final step is setting the threshold value. This is a value measured on the horizontal axis of the demodulated waveform. It tells the computer to consider anything above the threshold to be a logical 1, and everything below to be a 0. Set this to roughly the midpoint between 0 and the highest value of the waveform (you can zoom in on any part of the waveform by left-clicking and dragging a box; right-click to undo). Once you've got the threshold set correctly, you should then see a very clear digital waveform in red, transitioning from 0 to 1 and back again. You've now tuned to the signal and demodulated the radio waveform to recover the original digital signal.

You can stop the flowgraph when you've finished looking at the results.

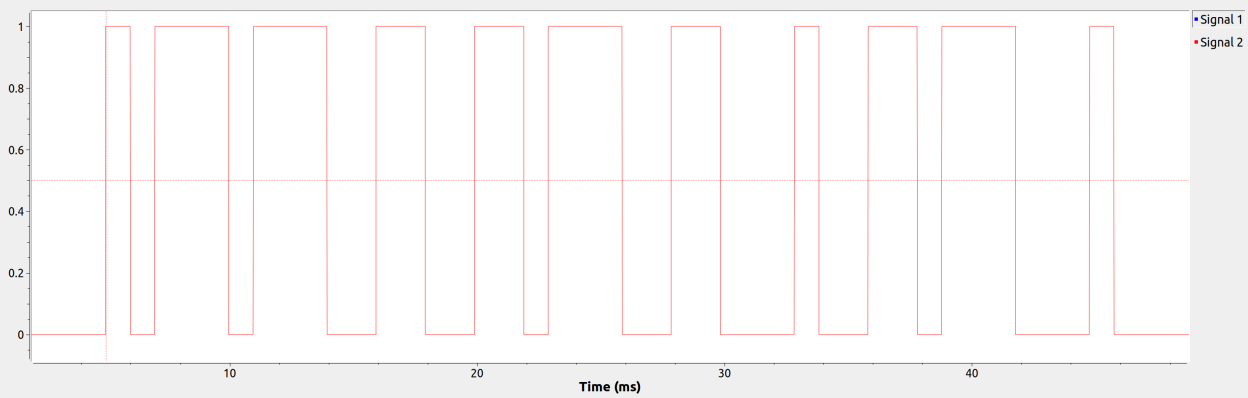
Solution 3

The signal is “located’ at 390 MHz, so set the frequency slider accordingly. The width of the signal is not much more than than a few kHz, but you’ll get a clearer view of your waveform if you set it a bit higher. Going all the way up to 30 kHz produces some really nice looking transitions.

Using the capture file we supplied you produces a demodulated waveform that transitions between 0 and roughly 0.05. The midpoint between these two values is 0.025, to which you should set your threshold.

Step 4 - Identify the Data

The next challenge is to read the ones and zeros in the waveform we obtained in the last step. This can be a bit like a puzzle. Before we tell you anything further, stop reading and just take a look at the waveform. You can close the previous flowgraph, open 04_id_the_bits/id_bits.grc, and run it to view the waveform again. Or you can just look at the image below:



See if you can “read” the ones and zeros in it by eyeballing it.

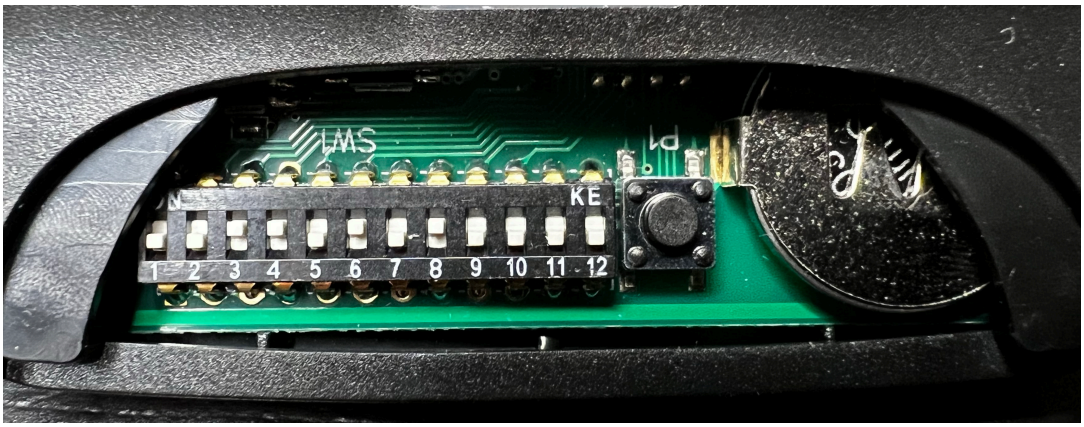
The key to decoding the digital data is to look for repeating patterns in the waveform. Usually these patterns will have the same width (or duration in time). This means most of the waveform will be made up of just two patterns that you’ll see over and over again.

I say “most” because you may also see another pattern that doesn’t actually represent data, but there for other reasons. If this third pattern is present in your waveform, it’s most likely at the beginning of a transmission. In this case it’s called the ***preamble***, and lets the receiver know when a new transmission is coming. In other cases, you might see this pattern at the end, telling the receiver when transmission is done - this is less common. Rarely, you may even see this third pattern in the middle of a waveform.

So back to waveform. See if you can find two main patterns that repeat throughout the signal. Remember, there may be a third used for purposes other than data.

(Please proceed to next page)

To help you figure out the patterns, we're going to let you look at the garage door remote itself and see how the switches are set:



These switch positions represent the ***address*** of the device, and the garage door won't respond to transmissions unless it's been programmed recognize this address. Accordingly, we'd expect to see some or all of this data in the waveform (this is a universal garage door remote, and not all switches are used in all protocols).

You may also find it helpful to convert the waveform into high and low levels based on the smallest unit of timing you observed earlier. To do this, you need to set two things: a symbol length and a preamble.

The symbol length is simply the measurement of the smallest time unit in the waveform. Enter this value into the Variable named symbol_len, on the right edge of your flowgraph. You can enter millisecond values with the same exponential notation we used for MHz, just with a negative exponent: for example, 10 milliseconds would be 10e-3.

Next, you'll need to look at the beginning of the waveform and see if there's a unique starting pattern, a preamble. Think in terms of this unit timing. For example, if a waveform went back and forth from low to high, every minimum unit of time, and it did so four times, you'd have a preamble of "01010101". What string of 0s and 1s could you use to uniquely represent the start of the garage door remote's transmission? Enter this string into the Variable named preamble_str.

Now when you run the flowgraph, you should see data scrolling by in the console window, that area below the graphical entry part of your GNU Radio Companion window. You may have to move the pop-up execution window of to the side to see it. The values scrolling by will be zeros and ones. These are not logical 0s and 1s, but you can see the repeating patterns that make up the 0s and 1s.

This can take a few minutes to figure out, so don't worry if it's not immediately obvious how to convert the waveform into binary data.

Solution 4

The minimum time between transitions is 1 millisecond, so the symbol_len value is 1e-3.

The two repeating patterns you see are:
- Two units low followed by two units high (pattern A)
- One unit low followed by three units high (pattern B)

But you also see a third pattern, at the beginning, the end, and one point in the middle:
- Three units low followed by one unit high (pattern C)

Thus, the signal looks like this:
CBBAABACBAC

Based on the physical switches in the remote (0011 0101 0000), it makes the most sense if:
B -> logic 0
A -> logic 1
C -> preamble pattern, ending pattern, and fixed value for the 7th address bit

They'd match up like this:
```
C B B A A B A C B A C
x 0 0 1 1 0 1 x 1 0 x (switch 7 and 10-12 unused)
```

You can also see these sequences in the console data (pattern A in Blue, pattern B in Green), and the special pattern C with no background):
0000: 00 01 01 01 00 01 01 01 00 00 01 01 00 00 01 01
0010: 00 01 01 01 00 00 01 01 00 00 00 01 00 00 01 01
0020: 00 01 01 01 00 00 00 01

NOTE: every once in a while the synchronization may glitch, so don't worry if one or two of the payloads deviate from what you see above

Step 5 - Take Over Original Door

Now that you understand how the remote's waveform is structured, you can build one of your own. Close your current flowgraph and open 05_tx_door_1/door1_tx.grc for a partially complete transmitter flowgraph. The flowgraph is set up to transmit 1s and 0s at a unit time of 1 millisecond. You just need to tell it what kind of waveform to send by providing the Vector Source block on the left with a Python list containing the 1s and 0s that will make up your transmission. If you haven't done any Python coding before, feel free to skip to the solution.

Just a few reminders on how list concatenation and list expansion work. The '+' operator will concatenate two lists together, and the '*' operator will created the specified number of copies of a list (technically, these are tuples below, but things work the same way). For example:
```
(0, 0, 1, 1) + (0, 1, 1, 1) = (0, 0, 1, 1, 0, 1, 1, 1)
2 * (0, 0, 1, 1) = (0, 0, 1, 1, 0, 0, 1, 1)
```

It's best to start at the lowest level and build up. Think about how you'd build the three basic sequences in 1 ms pieces, then enter them into the variable seq_one, seq_zero, and seq_x.

Next, come up with a Python expression for the Vector Source that will create 10 copies of the signal you want to send, with some dead air (not transmitting anything) between each payload. The existing variables in the flowgraph will help with this. When you run this flowgraph, you should see your transmission physically affect the garage door mock-up!

Solution 5

To create the three sequences, you want:
seq_one: (0, 0, 1, 1)
seq_zero: (0, 1, 1, 1)
seq_x: (0, 0, 0, 1)

To build the full waveform and send a burst of 10 transmissions with dead air, use the following in your Vector Source: 10 * (dead_air + payload)

Step 6 - Take Over New Door with Known Address

Now that you're actually sending bits and taking over the first system, it's time to extend the functionality of your transmitter to take over a second system. Assume that the switches on this second remote are set to the following:

1010 1101 0000

How would you modify the flowgraph in 06_tx_door_2/door2_tx.grc to produce this signal? The only thing you'll need to change is the payload variable.

When you're done with your modifications, run the flowgraph and see if you can control the second garage door!

Solution 6

The switch values map onto the waveform sequences as follows:

```
1010 1101 0000
|||| |||| ||||
ABAB AAxA B
```

After adding the special sequence to the beginning and end, this means the payload variable can be set to:

```
seq_x + seq_one + seq_zero + seq_one + seq_zero + seq_one + seq_one +
seq_x + seq_one + seq_zero + seq_x
```

Step 7 - Take Over Door with Unknown Address

Now imagine there's a third garage door, using the same make and model of remote control, but with an unknown address. How would you go about accessing such a door?

You can close GNU Radio Companion now by clicking the 'X' in the top right corner of the window. This will leave you at a terminal, from which you can type the following:
cd 07_tx_door3
gedit garage_door_brute.py

(Feel free to use the text editor of your choice here)

This will bring up a text editor containing a Python script. The script already does quite a bit, you just need to finish it. Starting at line 84, you'll see some instructions and two points between which you should insert your own code. Again, if you're unfamiliar with Python, feel free to jump to the solution file.

The first function you're given is called "build_payload" which takes an integer representing the switch positions, and builds a burst of payloads, just like you gave the Vector Source in the last two projects. The second function "tx_socket.send_raw_bytes" transmits the waveform you've built through your SDR.

One thing to note: the build_payload function assumes that the seventh bit will always be the "x" sequence, and skips over that bit. In other words, the codes don't go from 0 to 511 (9 bits), with an ignored bit in the middle, they simply go from 0 to 255 (8 bits). You're giving this function a 8-bit integer.

After you send each transmission, you should call the time.sleep function for 2 seconds so that there's enough time for the transmission to occur before moving on to the next one.

Solution 7

As you can see from the solution project, the key is iterating over all possible values from 0 to 0xFF and sending a transmission containing each.

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!