IoT Lab

Reverse Engineering of a radio protocol

Objectives:

- *Analysis of a raw SDR capture of a radio signal*
  - *Retrieving the frequency of the carrier wave*
  - *Analyzing the signal profile to make assumptions about the modulation type*
- *Analysis of the SDR capture with a hacking tool*
  - *Configuring a filter to retrieve bit streams*
  - *Analyzing the bit streams patterns to retrieve the encoding*
  - *Decoding the signal*
  - *Making assumptions about the protocol used by the device*
  - *Decoding the full signal pattern*
- *Signal encoding*
  - *Creating and generating a signal based on the previous analysis*

The story…

Our company wants to deploy some IoT devices to help employees in their daily lives and reduce costs. The purchasing department has identified a specific brand of products and would like you to participate in assessing its potential security risks.

You have been given an important mission! Analyze the radio protocols used by the devices and establish the hacking possibilities from an outsider's point of view.

The studied device is a remote control used by a mid-cost brand of Smart Home IoT Devices. The brand offers different types of devices: smart plugs, bell rings, remote switches, remote controls, …

According to the user manual, a pairing procedure is proposed when adding new devices to the connected environment, giving some sense of "security"…

Happy Hunting!

Part I: FFT analysis of the raw data using inspectrum

The goal of this part is to perform a first analysis of the captured signal to figure out its main characteristics and do a first decoding manually.

Based on the information found on the remote-control tag, a capture has been performed with an SDR device around a 433 MHz central frequency (frequency used might vary).

All the necessary files for your analysis are found in an archive named ***LabFiles.zip***.

The raw data are found in the file "***rawdata-B1-I***" file

- Open the raw data file with *inspectrum*
- Locate the recorded signal (a burst of at least 5 similar "long" sequences) and experiment with *inspectrum* to analyze it
- Indications:
    - What can you conclude about the frequency analysis?
    - Considering your answer to the previous question, what seems to be the type of modulation used?
    - Taking into account the fact that the modulated signal is a digital one, look at the signal profile and try to figure out how 0s and 1s are represented?
        - Tips:
            - You can add an amplitude plot to the FFT graph by right-clicking in the FFT window and then a threshold one
            - Understanding that the signal captured multiple bursts, try to identify what seems to "start" a burst.
            - Consider that this "synchronization" signal is not part of the digital signal
    - How long (in bits) is a full signal burst?
    - Manually decode the encoded binary sequence of a burst (tip: use cursors to trace your progress in the sequence)
- Make sure you validate your analysis with your professor.

Part II: Signal analysis automation using *URH*

Now that you understand how the remote control communicates, you are going to try to automate the decoding and then try to figure out the protocol used.

URH is a software specialized in radio signal analysis. It helps in decoding digital radio signals, identifying participants and protocols, generating signals and simulating protocols.

You are going to use it to automatically decode the previous signal, identify the encoding, decode the protocol and even generate your own commands.

The remote control you want to reverse engineer is composed of 3 rows of 2 switch-buttons. The switch-buttons, labelled respectively I and O, are used to switch on and switch off compatible IoT devices.

The two files located in the ***LabFiles*** directory contains recordings performed with an SDR device in different situations:

- "***B1-I***" is a command obtained by pressing the ON button of the first button row. It will be used to automate the decoding of the signal. It is a "pre-filtered" version of the raw signal used in the previous part

- "***FullRemote***" is a recording of multiple bursts obtained after pressing the different switch-buttons. It will be used to figure out the protocol used and identify the buttons and the actions to perform

Decoding the signal

First steps with *URH*

- Open the "***B1-I***" file in *URH*
- Check the *Modulation* is set to the one you found out in the previous part
- Edit the *Modulation* characteristics and change the pause threshold to 12 so as to be able to separate the synchronization sequences from the actual bit streams
- Click on "Autodetect *parameters*" to process the signal
    - You should obtain two types of "messages": the synchronization messages and the bit streams
- As you can see the bit streams don't make any sense so far and further analysis needs to be performed. You should however be able to understand why *URH* decodes it this way
- Click on the *Analysis* tab: this tab will provide you with a "digital" view of the signal
- Click on the "*Decoding*" menu and select the 3 dots "…". This should open a new window where you will be able to create your own decoding patterns based on standard ones

Transforming a raw signal into a valid bit stream

In this part, you are going to automate what you did manually with *inspectrum*.

- In the decoding window that you opened in the previous part:
    - Add an "*Invert*" function to "*Your Decoding*"; note that you can use the "*Test*" menu to see the result on the raw signal
    - Add a "*Morse Code*" found the "*Base Functions*" list to "*Your Decoding*". Adjust the parameters so that:
        - The Maximum length of 1-sequence for: Low (0) is set to 2
        - The Minimum length of 1-sequence for: High (1) is set to 7
        - The Number of 0s between 1-sequences is set to 1
    - Save the decoding pattern as: "*IoT Device*" and close the window
    - Back in the "*Analysis*" window, select your new Decoding pattern and apply to it the bit streams
    - Compare the obtained results with what you manually decoded in the first part.
    - You should be able to justify the Decoding method you just created

Decoding the signal

When looking at the global bit pattern obtained with the decoding method, it appears that there seems to be an unusual number 01 or 10 sequences when you usually expect to have longer sequences of consecutive 0s or 1s. A wild guess is that some sort of encoded was used to reduce transmission errors by encoding 0s and 1s with 01 or 10 transitions. Manchester encoding is often used in this situation.

Search how such an encoding works and modify your "*IoT Device*" decoder to include the new decoding pattern.

Checkpoint: if you decoded the sequence as expected, the hex value representing the burst should be: 0x422f0590.

Decoding the protocol

If the previous analysis is correct, we should now be able to apply our decoder to the "***FullRemote***" file and find some patterns to:

- Identify the actual switch-button
- Identify the ON/OFF command

In order to do that, you are going to use the comparison feature of *URH*:

- Open the "***FullRemote***" file
- Optional: to simplify the analysis, you can extract 6 bursts identifying the 6 different key pressed by using the menu options found when you select signal parts and right click on it
- Apply the proper modulation filter with the proper parameters
- In the *Analysis* window, only keep the bit streams (hide the synch messages) and apply your "*IoT Device*" decoder
- Use the comparison tool and try to identify where is/are located:
    - The button ID bits
    - The On/Off command

Part III: Bonus part - Generating your own commands

Now is time to see if you can generate your own command based on the decoded protocol.

- Open the "*B1-I*" file
- In the *Interpretation* tab, use the proper modulation & parameters
- In the *Analysis* tab, use your "*IoT Device*" decoder on the concerned bit streams
- Click on the *Generator* tab
- Drag & Drop the ***B1-I*** signal into the *Generated Data* area
- Click on *Generate* File.
  Should you have a connected SDR device, you could send the generated command over the air.
  Save it as ***Generated.B1-I.complex***
- In the *Generated Data* area, modify the command bit you identified earlier to send an OFF command
- Generate a new file. You will name this new file: ***Generated.B1-O.complex***

You are now going to check the generated signals are correctly decoded with *URH*.

- Close all files and open the two generated files
- Set the modulation parameters if necessary
- Click on the *Analysis* tab to check everything is as expected

Note: In a real-life situation, especially for this type of non-standard signal, you would have to be very precise in the way the signal is generated:

- You would have to precisely set the Samples / Symbol value, probably overriding the “autodetect” ones
- Retrieve the exact timings for 0s and 1s and write a program to exactly generate the signal