Project 5: Build a Sigfox application and launch a replay attack

AARJAV KOTHARI, [redacted], Univeristy of California Los Angeles
PRAGYA SHARMA, [redacted], Univeristy of California Los Angeles
MICHAEL SHERMAN, [redacted], Univeristy of California Los Angeles

Sigfox is an emerging LPWAN (low-power wide-area network) technology that serves long-range, power-constraint IoT devices. It is simple in its design and thus suitable for low-end IoT devices. It supports reliable, anti-jamming services as it employs diversity where each message is repeated for multiple times at different time and frequency. A Sigfox device does not need to establish connection with any base station; instead, any base station that captures the Sigfox data will process it. Therefore, it suffers from the packet replay attack: An attacker can capture a packet and replay it later (when certain condition is met). The receiver can falsely accept the replay from the attacker. In this project, we implement a Sigfox application and demonstrate a man-in-the-middle attack where an attacker node (USRP B210) captures and replays the Sigfox data.

Additional Key Words and Phrases: Low-power Wide Area Networks, Sigfox, GNU Radio, Man-in-the-middle

1 INTRODUCTION

In this project, we set up two devices to communicate with each other. We set up a plug-and-play Pycom FiPy [3] to communicate with the Sigfox base station. Once having connected the FiPy to a computer over USB and having downloaded the Pymakr extension for our preferred code editor, we can run code on the FiPy that attempts to communicate with local Sigfox applications over the RC2 Radio Configuration (RC2 is for the US and it operates in the 900MHz frequency band). One can perform several different actions of communication, namely the uplink and downlink. The uplink involves sending data to the Sigfox base station while the downlink involves receiving information from the Sigfox base station.

1.1 Sigfox Test Bed

The Sigfox application is a combination of the SDR Dongle hardware and the Sigfox Network Emulator (SNE) software solution. The SDR Dongle is a USB stick that can connect directly to your personal device and comes with an antenna to receive incoming messages. The SNE emulates a Sigfox base station and is not connected to the Sigfox backend.

In order to communicate with the two, we first had to retrieve the identity of the FiPy and register it with the Sigfox SNE. Once this connection is established and the SNE is running, we can ping our Sigfox base station by running the FiPy code. One important thing to note is that Sigfox uses Advanced Encryption Standard (AES) 128 public key encryption to perform authentication and data integrity checks [4]. Each Sigfox device has the option to enable or disable AES. Without authentication enabled, the SNE can still receive messages but cannot authenticate the sender and thus cannot test downlink messages. Additionally, you must enable the Pycom device to use this public key manually in the code.

Authors' addresses: Aarjav Kothari, 805526480, Univeristy of California Los Angeles, aarjavkothari23@ucla.edu; Pragya Sharma, 305033739, Univeristy of California Los Angeles, pragyasharma@ucla.edu; Michael Sherman, 605507996, Univeristy of California Los Angeles, msherman32@ucla.edu.

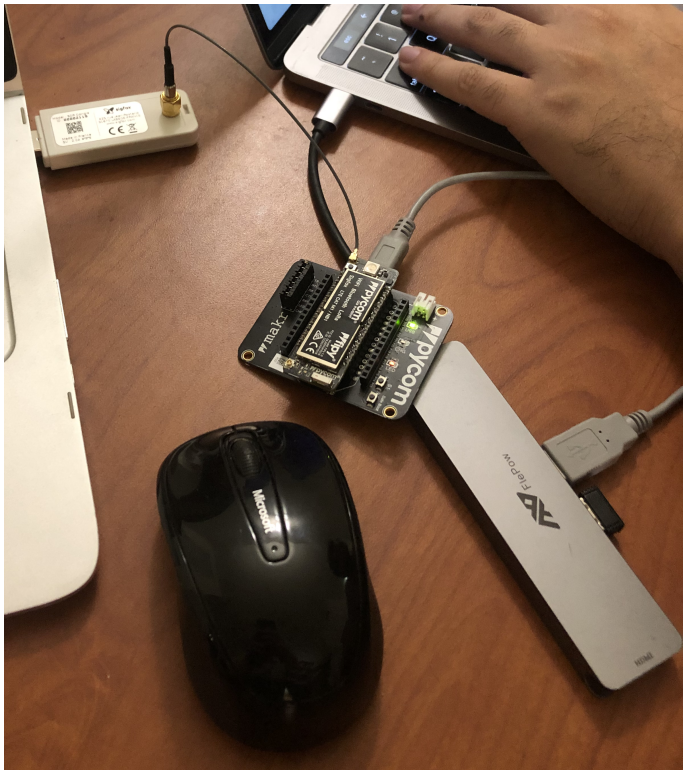

Fig. 1. The SigFox SDR Dongle (top left) and Pycom FiPy (middle) testbed

1.2 USRP Process

For the second part of the project to emulate an attacker node, we use USRP B210. Being the most widely used Software Defined Radio owing to its higher performance and wide frequency range (70Mhz to 6GHz), it is an industry-ready platform. To program the USRP, we make use of GNU Radio Companion [2]. The main benefit of the GUI comes from its easy plug-and-play and visualization environment. Figure 1 outlines the flow graph we plan on using to capture Sigfox communication between FiPy and the Base Station.

2 POSITIVE FINDINGS

We have been able to communicate successfully between the FiPy and the Sigfox Base Station over the air. We have found that the Sigfox works quicker outside rather than inside buildings and have always been able to get the uplink working properly (i.e. receiving bytes from the FiPy on the Sigfox base station). We have not been able to get the downlink to work locally. However, using the Sigfox backend instead of a local callback, we were able to successfully receive the downlink on the FiPy's side.

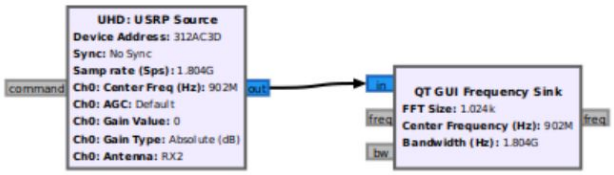

Fig. 2. USRP flow graph

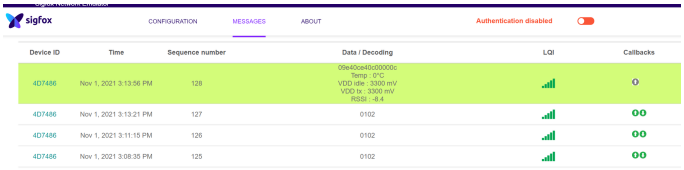

Fig. 3. Successful downlink response (highlighted in green) from the Sigfox Backend

We used our SDR dongle to create a communication with the FiPy device. Once the message is received by the SDR dongle, Sigfox creates a downlink message to the FiPy using a callback function. We setup our Sigfox to return a simple 8 Byte message "CAFEBABE" using a local host web application. The Sigfox backend software connects to our host and sends a payload to which we reply with a HTTP 200 OK response with the payload we need or a HTTP 204 No Content response with no message sent back.

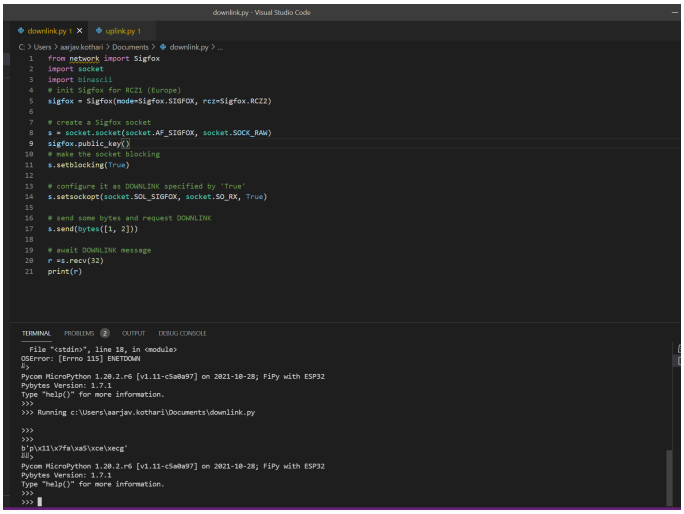

Fig. 4. Code snippet of python script running on Pycom FiPy to attempt to communicate bidirectionally with the Sigfox SDR Dongle

We also found out that google has a IoT cloud github setup with Sigfox . We found tutorials that explained how to ingest data from Sigfox devices and publish it to the Google cloud services along with device configuration management and service message logging. The Device configurations are stored in Firestore and logs in the Cloud logging. The implementation is set to be low cost, scalable and stateless[1].

3 TECHNICAL OBSTACLES

The FiPy device does not have a encryption state and does not share a public key with the Sigfox base station as stated on their website [3] [4]. This leads to unstable downlink connections. We have had issues where our message is acknowledged by the Sigfox base station but not retrieved by our FiPy. We cannot seem to figure out the solution of why we randomly cannot recieve the uplink. We hope to solve this by capturing our uplink on the USRP. Sigfox documentation suggests that there is a downlink limit of 4 messages per day which also could be why we have inconsistent results for the downlink [4].

Moreover, we have faced challenges in our attempts to capture the Sigfox communication between the FiPy and the Sigfox base station. The USRP device is not being recognized by the UHD libraries of the GNU Radio which prevents us from accessing and programming the radio on the SDR. This is what we are focusing our efforts on currently and hope to resolve it by the end of Week 8.

4 SELF-EVALUATION

We are on schedule. We believe we should be able to capture the signals on the USRP and successfully replay them by Week 10.

5 TIMELINE

Below is our expected timeline and task list for the remaining weeks:

- Week 8 - As mentioned earlier, we aim to resolve technical issues with USRP interface during this week and capture signal exchanges between the FiPy and Sigfox base station.
- Week 9 - We hope to start replaying the signal that is emitted by the FiPy device after capturing it on the USRP. If time permits, we can try to decode the signal and attempt to change the emitted signals as well.
- Week 10 - This week is reserved for final touches, if any, and preparing for the project demo.

6 ACKNOWLEDGEMENTS

We would like to extend our gratitude to Prof. Songwu Lu for giving us the opportunity to work on this project and our mentor, Mr. Zhaowei Tan for his continuous support and encouragement.

REFERENCES

[1] [n.d.]. CLOUD.GOOGLE.COM. https://cloud.google.com/community/tutorials/sigfox-gw. Accessed: 2021-10-21.
[2] [n.d.]. GNURADIO.COM. https://www.ettus.com/sdr-software/gnu-radio. Accessed: 2021-10-15.
[3] [n.d.]. PYCOM.COM. https://pycom.io/product/fipy. Accessed: 2021-09-15.
[4] [n.d.]. SIGFOX.COM. https://www.sigfox.com. Accessed: 2021-09-15.