Write Up: Poor Lifi 2.0

Corentin POUPRY
corentin.poupry@edu.esiee.fr
ESIEESPACE

Quentin CHAUVIN
quentin.chauvin@edu.esiee.fr
ESIEESPACE

Introduction

Poor Lifi 2.0 is a hardware challenge that was presented at the **Hackday** final in ESIEE Paris. It follows last year's challenge, Poor Lifi, which used the frequency modulation of a light to pass information, including the flag.

> In the depths of space, a research shuttle from the United Planets Organization received a distress signal from an unknown vessel. The captain of the rescue shuttle gathered his team and prepared for a rescue mission. They located the distressed vessel, but encountered a major problem: the vessel had been severely damaged and could no longer emit communication signals. However, Aria noticed a strange infrared light emanating from the vessel. She immediately realized it was a coded communication sent by the crew members who were seeking help.
>
> The flag format is : `HACKDAY{flag}`

Description

At the place of the challenge, you can found on a central table a stabilized power supply connected to an arduino and a big LED connected to a heat sink, letting us think that it could deliver a great power. A 2N2222 transistor was connected between a pin of the arduino and the LED.

On the challenger table was an arduino, jumpers cables and an infrared receiver module like this one:

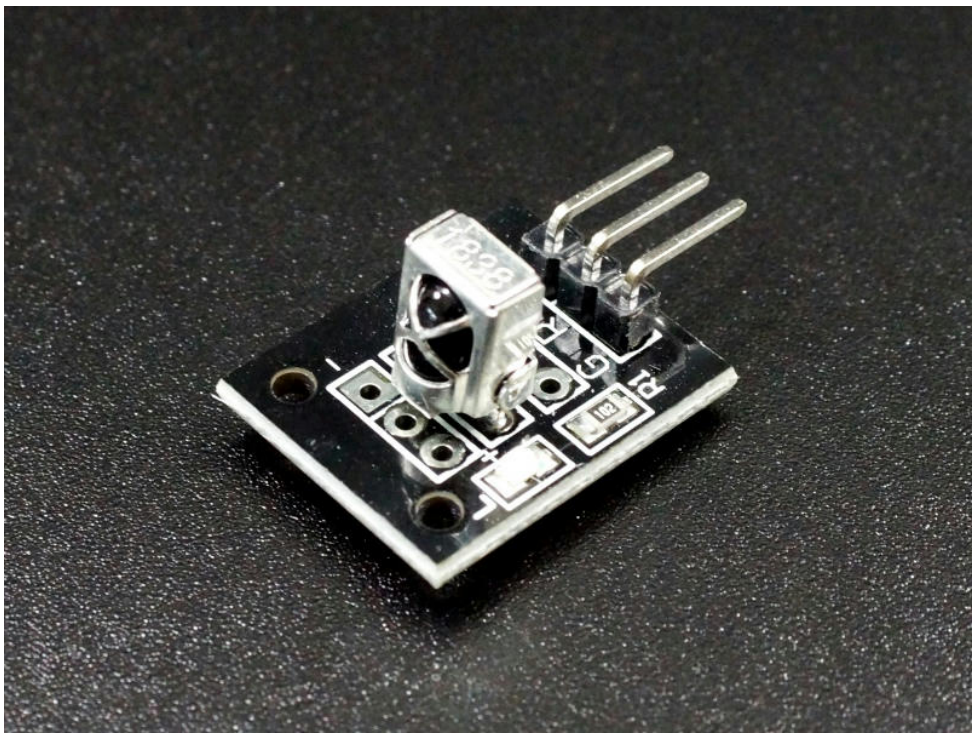

Figure 1: The IR receiver module used for Poor Lifi 2.0

Once the very simple assembly has been made and tested, we can start by letting the Arduino scan the different infrared signals using a library such as Arduino-IRremote with all the protocols activated: Arduino-IRremove repository provides a lot of usable examples.

```
Protocol=NEC Address=0x2 Command=0x34 Raw-Data=0xCB34FD02 32 bits LSB first
```

We then quickly notice that the transmitter uses the NEC IR transmission protocol. If we dump the various frames obtained, we obtain the following :

Command	Raw Data
0x27	0x50
0x27	0x50
0x27	0x50
0x2F	0x49
0x2F	0x49
0x6	0x59
0x6	0x59
0x6	0x59
0x6	0x59
0x0	0x48
0x0	0x48
0x6	0x59
0x6	0x59
0x6	0x59

Table 1: Dump from the IR receiver

And it goes on and on, with different values. One of the first things we can observe is that certain values are sent several times in successive frames (looping is typical of infrared transmissions, to ensure that the receiver doesn't miss a frame), but also that certain frames are sent afain with the same `Command` and identical `Raw Data` (for example, `Command 0x6`).

Note that the `Commands` seem to be an index (some `Command` are 0x0 in the dump and they are all generally less than 50 or `0x32`). `Raw Data` seems to contain higher figures, so we can try a naive translation using an ASCII table.

Command	Raw Data	ASCII
0x27	0x50	P
0x27	0x50	P
0x2F	0x49	I
0x2F	0x49	I
0x6	0x59	Y
0x6	0x59	Y
0x6	0x59	Y
0x6	0x59	Y
0x0	0x48	H
0x0	0x48	H
0x6	0x59	Y
0x6	0x59	Y
0x6	0x59	Y

Table 2: Dump translation using ASCII

Having the letter `H` and `Y` is very suspicious and shows us that we are on the right track, knowing that the flag is given in the form `HACKDAY{FLAG}`. It makes sense to try to order them using Command as a 0-indexed array index.

I advise you to write a program on your Arduino that sends `Command` and `Raw Data` via serial link, and a second program on your PC that assembles the flag as the data is sent from the Arduino. This way, if the Arduino ever crashes, you'll only have to reset it without losing the already assembled flag!

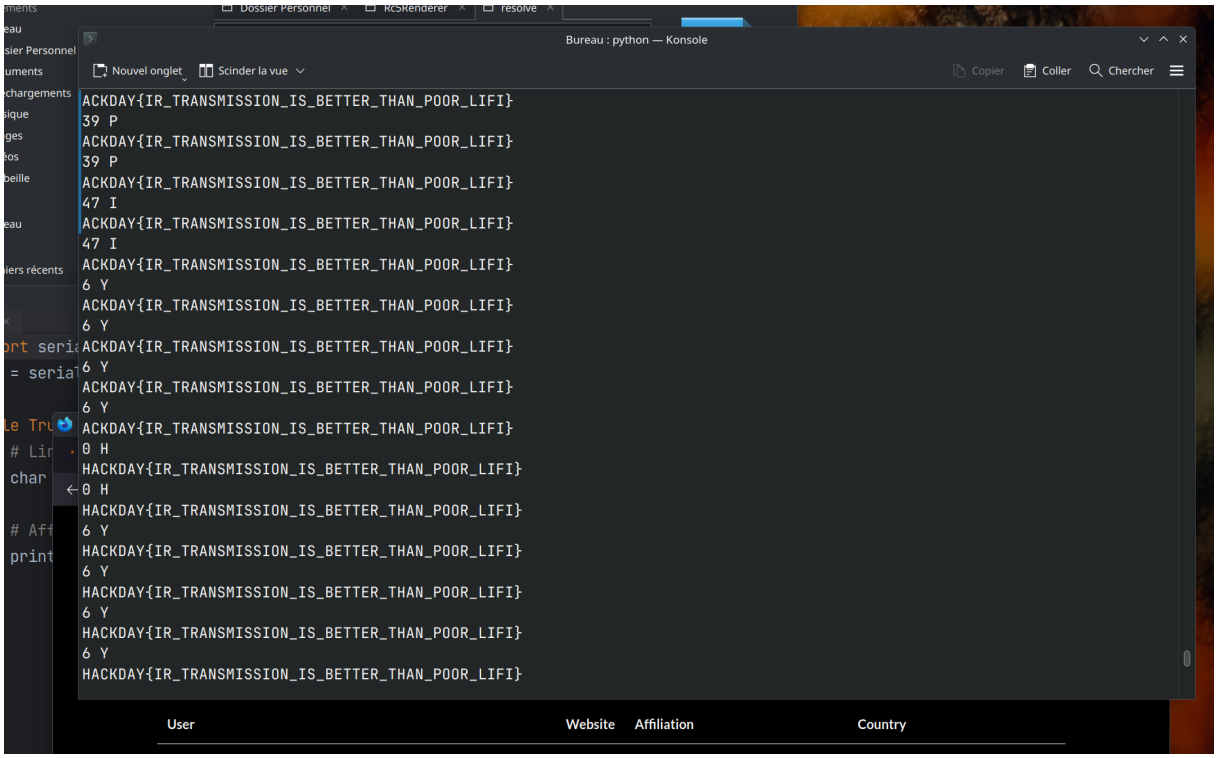

Figure 2: Flag assembly on computer

In this way, we obtain the flag `HACKDAY{IR_TRANSMISSION_IS_BETTER_THAN_POOR_LIFI}`.