Internship Project Documentation

Giuseppe Cervone
giuseppe.cervone@edu.unife.it

1 Introduction

The aim of the project is to develop what in cybersecurity and IoT is called a data diode, a unidirectional communication device for data exchange. The importance behind this project is that industrial data diodes are expensive devices, and implementing a high amount of them like for this specific use case is a very costly move, but with software implementations (firewalls) or hardware implementations (serial or optical communication built to be unidirectional) it is possible to have cheaper data diodes that are almost as functional for most use case scenario. The initial setup will include two Raspberry Pi model 3 with a specific configuration which we will highlight later, and a TTL-232R-3V3 cable without the RX pin being plugged in one of the raspberries.

2 Devices

2.1 Raspberry Pi

As far as the Raspberry Pi models, I have chosen to use the Raspberry Pi 3 as it's a nice midway point between performance and cost. This particular machine should be able to handle the data size we are planning to share (around 1Gb/day), at a cost that is much lower than that of a Raspberry Pi 4. The machines are configured as follows:

- Raspberry Pi 3 Model B Ver1.2:
  - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
  - 1GB RAM
  - BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
  - 100 Base Ethernet
  - 40-pin extended GPIO
  - 4 USB 2 ports
  - 4 Pole stereo output and composite video port
  - Full size HDMI
  - CSI camera port for connecting a Raspberry Pi camera
  - DSI display port for connecting a Raspberry Pi touchscreen display
  - Micro SD port for loading your operating system and storing data
  - Upgraded switched Micro USB power source up to 2.5A
  - Raspberry Pi OS Lite (latest version)

  - 16GB microSD card

Using the RPI-Imager tool I installed the latest Raspberry PI OS image. Upon first boot up, remember that raspberry pi login is username: pi, password: raspberry. As Raspberry Pi OS Lite is an OS without a desktop environment, we don't have a systems setting manager with a GUI, luckily we can use raspi-config to enable a series of features useful for testing software. We initialize raspi-config by using the command sudo raspi-config and then changing these specific settings:

- In the system options:
  - Enable WIFI for ssh purposes.
  - Change username and password if needed.
- In the interface options:
  - Enable SSH.
  - Enable Serial interface, remembering to disable login shell but enable serial interface.
- Run sudo apt-get upgrade to make sure all packages are up to date before using raspi-config.
- Disable bluetooth:
  - We won't be using bluetooth for the project, so we disable it, making the final product more secure.
  - We disable bluetooth by adding 'dtoverlay=disable-bt' in /boot/config.txt.
- Troubleshooting
  - Check that UART is enabled in /boot/config.txt by adding 'enable_UART=1'.
  - Check that serial console is disabled by removing "console=serial0,115200" (or "console=ttyS0,115200") in /boot/cmdline.txt.
  - Check for permissions in the dialout group.
  - Check that /dev/serial0 doesn't have a getty console running on it. In case it does, it can be disabled by using the commands: sudo systemctl stop serial-getty@ttyS0.service and sudo systemctl disable serial-getty@ttyS0.service.
- Additional software:
  - Run sudo apt-get install python3 to install Python in it's latest version.
  - Run sudo apt-get install python3-pip to install Python's package manager.
  - Run pip install pyserial to install the serial library that we will be using to have the machines comunicate.

2.2 Serial cable

As mentioned earlier, the serial communication will go through a TTL-232R-3V3 cable, which is USB to serial, with +3.3V TTL levels UART signals. The cable has 5-pins on one end, and USB on the other. Using GPIO14 pin on the RPi we can trasmit data, using GPIO15 we receive data. As previously mentioned, the connection with GPIO15 can be removed after the initial tests are over, as the communication will be one-way. In appendix 1, there is an extract from the the documentation of the cable, and a pinout image showing the correct way to plug in the cable in the correct pins, make sure that GND pin is in any of the ground pins available on the raspberry GPIO, and most of all remember that TX and RX are from the machine's point of view, thus they have to be plugged in the opposite way (GPIO14 with yellow and GPIO15 orange). With the cable plugged in we run the command ls /dev/tty* on both raspberry machines so we have an idea of what port we are using on each raspberry in the serial communication (For the port that's sending data it should be /dev/ttyS0, while for the other port it should be USB0). Raspian will always map the serial port to /dev/serial0, so it is suggested to use the alias in programming. In appendix 2 is an image of how the two machines should look like once plugged in, with an idea of how they interact with the network.

3 Program versions

3.1 Diagnostic test

The objective of this test is to see if the two RPi are connected correctly. In appendix 3 we have example code and example output that can be used to troubleshoot and check that the cable has been plugged in correctly. In case this program won't run correctly, refer back to the troubleshooting section earlier. Make sure to run the code sndtest.py on the machine where the GPIO pins are being used, and recvtest.py has to be run on the machine where the USB is plugged in. It's suggested you start by running recvtest.py using the command python3 recvtest.py, and then run python3 sndtest.py.

3.2 Data diode 0.1

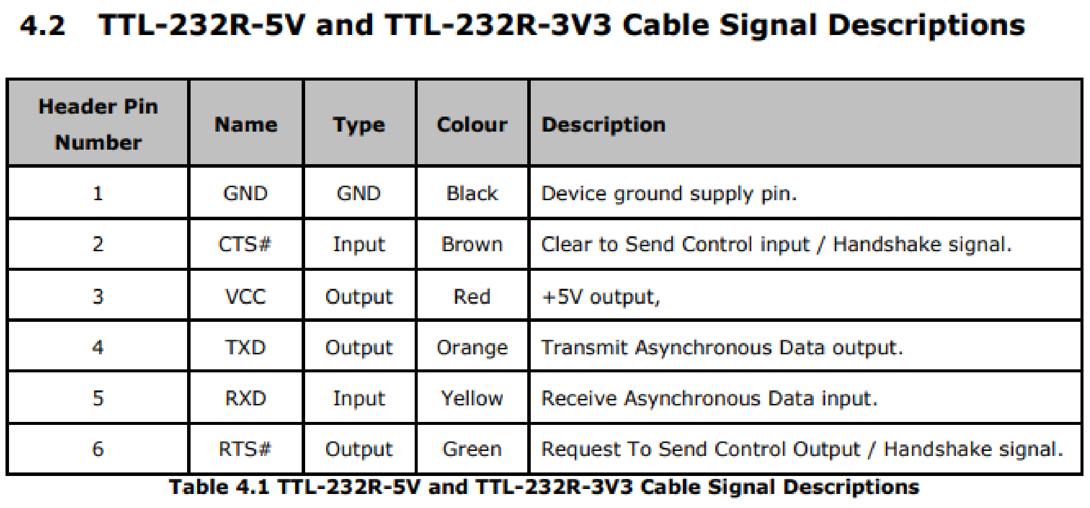
In this initial implementation we switch from sending text to sending an actual file via serial communication. The improvements from this version to the next will be improved baudrate, meaning faster file transfer. In the next version SHA256 checksum will also be implemented to make sure file transfer is working correctly. The last improvement will be to include a handshake system via CTS/RTS signals. The code can be found in the appendix 4.

3.3 Data diode 0.2

This version of the program has SHA256 integration to validate file transfer and improved transfer speeds thanks to a much better baudrate. The limitations with this version of the program is that because of limitations of USB2.0 and limitations of the UART communication protocol, 115200 baudrate seems to be the maximum baudrate without introducing errors. Transfer speeds are improved from 0.1 but they have be improved by changing the communication protocol, a comparative of baudrates can be seen in the appendix section 5, with the code example.

4 Appendix

4.1 Appendix 1: Documentation extract and pinout raspberry



4.2 TTL-232R-5V and TTL-232R-3V3 Cable Signal Descriptions

Header Pin Number	Name	Type	Colour	Description
1	GND	GND	Black	Device ground supply pin.
2	CTS#	Input	Brown	Clear to Send Control input / Handshake signal.
3	VCC	Output	Red	+5V output,
4	TXD	Output	Orange	Transmit Asynchronous Data output.
5	RXD	Input	Yellow	Receive Asynchronous Data input.
6	RTS#	Output	Green	Request To Send Control Output / Handshake signal.

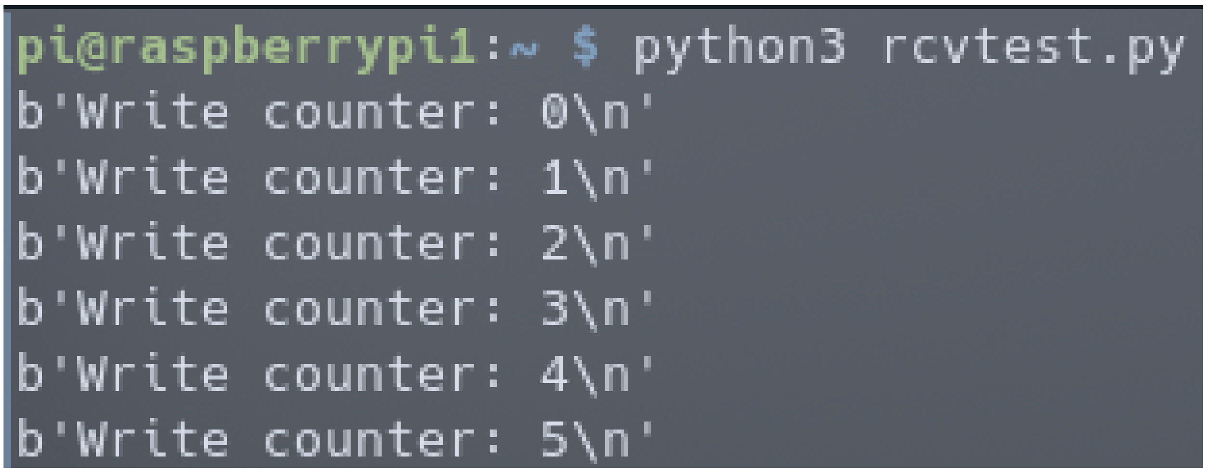
Table 4.1 TTL-232R-5V and TTL-232R-3V3 Cable Signal Descriptions

Missing raspberry pinout

4.2 Appendix 2: Network map

Missing network map

4.3 Appendix 3: Diagnostic output and code example



```
pi@raspberrypi1:~ $ python3 rcvtest.py
b'Write counter: 0\n'
b'Write counter: 1\n'
b'Write counter: 2\n'
b'Write counter: 3\n'
b'Write counter: 4\n'
b'Write counter: 5\n'
```

Listing 1: sndtest.py

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
        port='/dev/serial0',
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)
counter=0

while True:
        ser.write(b'Write␣counter:␣%d␣\n'%(counter))
        time.sleep(1)
        counter += 1
```

Listing 2: rcvtest.py

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
        port='/dev/ttyUSB0',
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)

while True:
        x=ser.readline()
        print(x)
```

4.4 Appendix 4: Data diode 0.1 code example

Listing 3: sndfile0.1.py

```
import serial
import time
import os
from hurry.filesize import size,alternative

ser = serial.Serial(
        port = "/dev/serial0",
        baudrate = 9600,
        parity = serial.PARITY_NONE,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS,
        timeout=1
        )
#Initialize serial port

filesize = size(os.path.getsize('log.txt'), system=alternative) #gets filesize in

print("Starting␣transfer...␣Time␣started")
start_time= time.time()
```

```
#Here we open the file , and keep on reading until there's lines , we then send <<EO

f = open("log.txt","rb")
line = f.readline()
while line:
    ser.write(line)
    line = f.readline()

ser.write(b"<<EOF>>")

f.close()

end_time = time.time() - start_time

print('Transfer␣completed...␣Filesize:',filesize,'.␣Time:',end_time,'seconds.')
```

Listing 4: rcvfile0.1.py

```
import serial
import time

ser = serial.Serial(
        port = "/dev/ttyUSB0",
        baudrate = 9600,
        parity = serial.PARITY_NONE,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS,
        timeout = 1
        )

eof = '<<EOF>>'

f = open("log.txt", "wb")

x = ser.readline()

while x != eof:
    f.write(x)
    x = ser.readline()
f.close()
```

4.5 Appendix 5: Data diode 0.2 code example and transfer speeds limitations

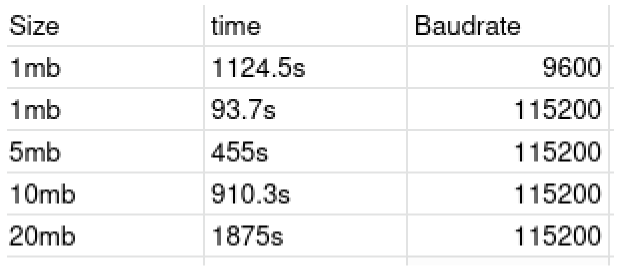
Listing 5: sndfile0.2.py

```
import serial
import time
import os
from hurry.filesize import size,alternative
import hashlib

ser = serial.Serial(
        port = "/dev/serial0",
        baudrate = 115200,
        parity = serial.PARITY_NONE,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS,
        timeout=1
        )
#Initialize serial port

filesize = size(os.path.getsize('log.txt'), system=alternative) #gets filesize in 

sha256_hash = hashlib.sha256() #we will calculate the SHA256 of the file to verify

f = open("log.txt","rb")
print("Starting␣transfer...␣Time␣started")
start_time= time.time()
#Here we open the file, and keep on reading until there's lines, we then send <<EO
ser.flushOutput()
line = f.readline()
while line:
    sha256_hash.update(line)
    ser.write(line)
    line = f.readline()
end_time = time.time() - start_time
f.close()
ser.write(b"<<EOF>>\n")

sha256_hash = sha256_hash.digest() #digest transforms into string
ser.write(sha256_hash)


print('Transfer␣completed...␣Filesize:',filesize,'.␣Time:',end_time,'seconds.')
```

Listing 6: rcvfile0.2.py

```
import serial
import time
import hashlib

ser = serial.Serial(
        port = "/dev/ttyUSB0",
        baudrate = 115200,
        parity = serial.PARITY_NONE,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS,
        timeout = 1
        )
#initialize serial port

eof = b"<<EOF>>\n"
#define EOF, signals to the receiver that the file is over
```

```
sha256_hash = hashlib.sha256() #we will calculate the SHA256 of the file to verify

f = open("log.txt", "wb") #open file to write into
ser.flushInput()
x = ser.readline()
while x != eof: #until file comes, decode, if x=eof then check if file transfered
    sha256_hash.update(x) #decoding is done in bytes, so we make sha before making
    f.write(x)
    x = ser.readline()
f.close()

sha256_send = ser.read(32)
if sha256_send != sha256_hash.digest():
    print("File␣Transfer␣failed")
else:
    print("File␣Transfer␣success")
```



Size	time	Baudrate
1mb	1124.5s	9600
1mb	93.7s	115200
5mb	455s	115200
10mb	910.3s	115200
20mb	1875s	115200