Blog Post

# Introduction

At Queens University Belfast there is a module called “Computer Science Challenges” which I am enrolled on. This module sets Year 1 students a momentous computing task for their experience level and then sees how far we get along. The main goal is not to complete the task but to create things of value which will be useful to people.

My task was:

“Create **a raspberry pi, npcap and wireshark** based **packet sniffing network monitor** that can be attached to **wired and wireless networks** and **passively monitor network communications**. Time permitting explore how anomalous transmissions can be detected using machine learning based modelling of normal network behaviour. This project also requires an analysis of the data protection and privacy implications of the technology.”

# QEMU & Linux

This task intimidated me. I mean what would your response be to a paragraph in which you don’t know what half the words mean! Anyway, the first step I took to tackle this problem was to dual boot Linux onto my laptop. Why?, Well I knew from my (very limited) experience of using a raspberry pi is that they are very slow. An emulator would be much easier to work with. Enter QEMU (<https://www.qemu.org/>). QEMU is a generic and open source machine emulator and virtualizer, perfect for emulating a raspberry pi. Now granted QEMU can be run on a Windows operating system however there is performance loss running on Windows. I manage to install the Raspbian onto the QEMU, learning a variety of useful Linux commands. I had not used Linux before so I found this very rewarding to get working. Now on my machine the emulator still ran very slowly. This is because raspberry pis run on an ARM processor whilst my machine ran on an Intel x86. This difference in instruction sets meant that the instructions themselves needed to be translated from Intel x86 to ARM. You could imagine that the amount of instructions needed at one time would be quite high, so most processing resources were spent on this conversion. The work setting up this emulator was all for nothing upon the discovery of what NPCap was…

# NPCap

Npcap is the Nmap Project's packet sniffing (and sending) library for **Windows**. It is based on the discontinued WinPcap library, but with improved speed, portability, security, and efficiency. NPCap is dependent on the Windows operating system, thus I couldn’t run it on my raspberry pi, let alone my Linux emulated raspberry pi. I did some digging ( <https://github.com/nmap/nmap/issues/1590> , <https://www.raspberrypi.org/forums/viewtopic.php?t=143622)>) which reaffirmed that I could not use that package. This vexed me massively, as learning how to use NPCap took a long time, as someone who has not done c programming before. I even created a how-to guide for it, in order to help new users (INSERT LINK TO HOW-TO GUIDE HERE).

# TShark

What was I to do? With no packet sniffing package, how was I to progress? After much deliberation I decided to use tshark instead. Tshark is the terminal version of Wireshark, so it adhered to the project spec. It also works on most operating systems, including Raspbian.

I found using tshark to be a very enjoyable experience. After getting use to the syntax, I found myself reading and writing packet files from both wired and wireless networks. Since the GUI version, Wireshark, is much more popular than the terminal based version, there was limited learning resources. I created a video tutorial giving an overview of how tshark works (<https://www.youtube.com/watch?v=NvKiyHCRXrU&t=3s>). You may ask why I would use tshark over Wireshark for this project. The answer is that Wireshark, being GUI based requires a lot more computing resources than tshark.

# Setting Up Raspberry Pi

At this stage I assembled my raspberry pi. As this device was for **wired** and **wireless** networks, I was provided with a usb wireless adapter which allowed the raspberry pi to connect to the internet.

Here is a list of what peripherals I used:

* Raspberry Pi 3 (Running Raspbian 4.19 Buster)
* Monitor
* Mouse
* Keyboard
* Wi-Fi Adapter

Computing resources are scare on a raspberry pi, so any saving you can accrue A screen shot of a computer

Description automatically generatedis for the best.

Now with the packet files being produced, I now needed a way to send them to my laptop in an efficient manner.

# File Sending

I tried to use WinSCP to send files to and from my raspberry pi. WinSCP is a free SFTP,SCP,S3 and FTP client application. After watching a tutorial on the subject, I managed to link my laptop and the raspberry pi with an SSH connection using the wireless network. I was now able to **manually** send files to and from my laptop and the raspberry pi. This was good start, however I needed to **automatically** send the files between the two machines. This would involve setting up an SFTP server. I looked feverously for a good tutorial on the subject matter, but unfortunately, I could not find one. That process took about 2 days, due to the constant trying and restarting of different methods. Eventually I gave up on this particular approach and decided to do use something else…

Flask is a micro web framework written in Python. It is used to build simple websites and act as a webserver. How could this package be of use in this situation? I set up a webserver using Flask, which allowed me to then setup a URL which would, when visited, would automatically download a selected file. I ran this flask server on my raspberry pi and set the desired file to be a packet file. It worked perfectly… after a few iterations of the initialisation command. I was over the moon that I had found a solution to this problem as now I would be able to have a constant stream of packet files being sent to my laptop for processing.

A screenshot of a cell phone

Description automatically generated

In order to receive the file, I had to use another library, Pycurl, which is a Libcurl wrapper for python. The way it works is that Pycrul receives the file as a byte string. Pycrul then writes this byte string as a file with the specified extension.

A screenshot of a cell phone

Description automatically generated

# Processing (Cleaning) Data

Speaking of processing, this is what I moved onto next. My goal at this stage was to do a “proof of concept” for my machine learning model as I had no prior experience with machine learning and wanted something relatively simple to start. I decided that my machine learning model would determine if a packet was part of a DDOS or not. PyShark is a python wrapper for tshark. This is how I “read” my .pcap files into my program. I was very impressed by PyShark’s capabilities. I made a beginner’s guide to this library aswell(https://www.youtube.com/watch?v=dL3X4IAZVlA)

In order to learn the basics of machine learning, I watched a couple fast.ai(<https://www.fast.ai/>) lectures for their “Machine Learning for Coders” course. These lectures were great as they immediately told me how to implement a machine learning algorithm. Note that in the lectures they use a RandomForestRegressor but for my problem I needed a RandomForestClassifier. This is because I wanted to classify my packets into DDOS packets or normal packets.

Cleaning the data for this project proved quite tricky. In the fast.ai lectures it is made explicitly clear that all data must be in numerical form for the random forest to work. This means that data such as IP Addresses need to be treated as categories and assigned their own corresponding code. Also, certain protocols such as ICMP do not have port addresses so accounting for these protocols was also something to look out for.

A close up of text on a black background

Description automatically generated

Fast.ai recommends using pandas, which is a high-level data manipulation package for python, to manipulate data. This package, though extremely useful, did have some syntax and commands which were not very intuitive. Still I managed to clean the data for use in the model.

# Datasets

In order to train and test my model I had to source a few datasets for my project. This took a while as there are not many publicily available datasets of useful packet data files. Here is the list of useful datasets I found:

* Netresec https://www.netresec.com/?page=PcapFiles is an independent software vendor with focus on the network security field. They also maintain a dataset of pcap files, some of which come from honey pots.
* The paper “An empirical comparison of botnet detection methods”

<https://www.sciencedirect.com/science/article/pii/S0167404814000923>

hits a lot of the same themes that my project has.

The scholars in this paper created their own public dataset: download link here https://www.stratosphereips.org/datasets-ctu13.

* University of New Brunswick <https://www.unb.ca/cic/datasets/ddos-2019.html> has many different pcap datasets.

# Current Status & Possible Improvements

I am now currently debugging the prediction method for my model. Once this is complete, I can test my model.

This project could be expanded and improved in many different ways:

* Expanding the threat detection aspect of the model, so that it can detect more than just DDOS attacks.
* Trying different machine learning algorithms other than Random Forests
* Expand the project to give diagnostic information to help solve the problem/threat detected.
* Expand the project so that the device can disconnect the problem computer from the network.