WiFi-based Presence Detector - Part 1

WiFi · MAC Address · ARP Scan · CronTab

Introduction

In this lab you will use a WiFi local network and a Raspberry Pi/SenseHAT to build a device presence detector. You will scan the WiFi network for connected devices and identify them using their MAC address, the unique identifiers that devices use when communicating in a WiFi network. An application of this is to check who is home/in the office/lab (and not home) by checking if a person's smartphone has connected to the WiFi.

Equipment

- Raspberry pi 3 and SD Card installed with Raspbian
- SenseHAT
- A Wifi network with internet access
- OPTIONAL: Screen with HDMI, Keyboard, Mouse.

arp-scan

arp-scan is a fast ARP packet scanner that can find all active IPv4 devices on a network. Devices cannot hide from ARP packets like they can hide from Ping and it should detect all devices including those with firewalls.

To install arp-scan, connect to your RPi 3 in your preferred manner (e.g. headless or screen-based)

Open a terminal window on the Raspberry Pi and enter the following command:

```
sudo apt-get update
sudo apt-get install arp-scan
```

Once installed, check that it's working correctly by entering the command on the Raspberry Pi: `sudo arp-scan -l` . This will list all devices on your local network that responded . You should see a list of devices and corresponding IP and MAC addresses on your local network. It may take a moment to load if you are on a large network:

```
pi@sensePi:~ $ sudo arp-scan -l
Interface: wlan0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/)
192.168.1.1     c8:0e:14:46:c2:c1       (Unknown)
192.168.1.43    34:e6:d7:06:ef:6f       (Unknown)
192.168.1.63    a0:63:91:30:c5:9b       (Unknown)
192.168.1.20    b0:e1:7e:09:9c:fb       (Unknown)
192.168.1.55    00:22:61:e2:a0:50       Frontier Silicon Ltd
192.168.1.65    64:b5:c6:52:3a:85       (Unknown)
192.168.1.254   00:1d:7e:27:b8:04       Cisco-Linksys, LLC
192.168.1.56    d4:28:d5:37:7e:a2       (Unknown)

9 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9: 256 hosts scanned in 4.048 seconds (63.24 hosts/sec). 8 responded
pi@sensePi:~ $
```

arp scan

- Question: If you had to write a program to replicate the list function in arp-scan, how would you do it?

Presence detection

ARP-Scan lists all devices (if any) connected your local network at the time the scan was executed. By scanning the local network for certain devices' MAC addresses, we can detect their 'presence' on the network. Furthermore, if connected by WiFi, we can deduce that the physical device itself is within the range of the WiFi access point (e.g. at home).

- There are various ways of finding a particular device MAC address and a quick internet search will soon let you know how to find the MAC address for the WiFi interface on your Smartphone.
- Find and record the MAC address of your smartphones Wifi interface (or if you're not using a Smarthone, any other device on the Wifi Network for now)
- Check for the presence of the device by doing a 'grep' on the device list returned by apr-scan:

```
sudo arp-scan -l | grep YOUR_DEVICE_MAC
```

```
pi@sensePi:~ $ sudo arp-scan -l | grep d4:28:d5:37:7e:a2
192.168.1.56    d4:28:d5:37:7e:a2       (Unknown)
```

arp scan

If your device was found, the command will output its address info. If nothing appears, make sure that it's connected to the same local WiFi network as your RPi. Smart devices are fairly energy efficient so you may also need to 'wake up' your device, as it may drop the WiFi connection if left idle for too long.

Now that we have a mechanism to detect known devices on the local network, we can write a short python program to get the RPi/SenseHAT to indicate the presence/absence of a device.

Presence Detector Script

Using `arp-scan`, you will now write a script that:

- gets a list of connected devices to the local network
- gets a list of "known devices"
- echoes(prints) the intersection of connected devices and known devices

Put simply, the progran will find the list of known devices that are connected to the network.

Create the Search Script

- On the RPi, open a terminal window (connect either by SSH or Screen/Keyboard), and, in your home directory, create a new directory called `presence`:

```
pi@sensePi:~ $ mkdir presence
pi@sensePi:~ $ cd presence
pi@sensePi:~/presence $
```

make directory

In this directory, create a new file in called `presence-detect.sh` and enter the following code:

```
#! /bin/bash

# presence-detect.sh
# searches for the MAC address of known devices

# do arp_scan to get connected mac addresses
connectedDevices=$(sudo arp-scan -l)

knownDevices=("d4:28:d5:37:7e:a2")

for device in "${knownDevices[@]}"
do
   if [[ "$connectedDevices" = *"$device"* ]]; then
      echo "$device is present!"
   else
      echo "$device is NOT present!"
   fi
done
```

- Save the file and change the files permissions to make it executable:

```
chmod +x presence-detect.sh
```

- Test the script by running it at the command line (./presence-detect.sh) .

```
pi@sensePi:~/presence $ ./presence-detect.sh
d4:28:d5:37:7e:a2 is NOT present!
pi@sensePi:~/presence $
```

presence detector

The script gets the result of `arp-scan` and assigns it to connectedDevices. It then searches `connectedDevices` for each MAC addresses contained in the `knownDevices` array (the above script only contains my phones MAC address).
The script should return indicating that the device was not detected (otherwise I want my phone back!).

- Now update the script and add another device in the `knownDevices` list that you can connect/disconnect easily. Perhaps add in your smartphones address.

```
knownDevices=("d4:28:d5:37:7e:a2" "xx:xx:xx:xx:xx:xx")
```

Now run the script again. Make sure you get a result that has both outcomes (present and not present).

Python ARP-Scan

In order to access easily higher order functions on the RPi such as SenseHAT and messaging protocols, we'll now switch to using Python, a good general purpose programming library that's already installed on the RPi,

Scanning for MAC addresses with Python

We can call the `arp-scan` program from a Python program using the `subprocess` library.

- In the `presence` directory you created earlier, create a new file called `presence-detector.py` with the following content:

```
#!/usr/bin/env python
#coding=utf-8

import subprocess

def arp_scan():
    output = subprocess.check_output("sudo arp-scan -l", shell=True)
    print output

arp_scan()
```

- Run the program by typing `python presence-detector.py` on the command prompt. You should see the `arp-scan` output printed on the console similar to the following:

```
pi@sensePi:~/presence $ python presence-detect.py
Interface: wlan0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/)
192.168.1.1     c8:0e:14:46:c2:c1       (Unknown)
192.168.1.43    34:e6:d7:06:ef:6f       (Unknown)
192.168.1.24    84:d6:d0:77:6f:60       (Unknown)
192.168.1.63    a0:63:91:30:c5:9b       (Unknown)
192.168.1.55    00:22:61:e2:a0:50       Frontier Silicon Ltd
192.168.1.254   00:1d:7e:27:b8:04       Cisco-Linksys, LLC

6 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9: 256 hosts scanned in 3.840 seconds (66.67 hosts/sec). 6 responded
```

arp scan with python

Using this program we can now get at the MAC address list programatically.

- To search the output for a particular MAC addresses, lets introduce two lists into our program, device owner names (`names`) and corresponding MAC addresses (`macs`). Notice the order of the names and devices correlate(i.e. Frank's device MAC address is `"d4:28:d5:37:7e:a2"`). **As before, you should add a name and device that is present on the local network.**

```
#!/usr/bin/env python
#coding=utf-8

import subprocess

#Names of device owners
names = ["Frank","Someone Else"]

# MAC addresses of devices
macs = ["d4:28:d5:37:7e:a2","xx:xx:xx:xx:xx:xx"]

def arp_scan():
    output = subprocess.check_output("sudo arp-scan -l", shell=True)
    for i in range(len(names)):
        if macs[i] in output:
            print(names[i] + "'s device is present")
        else:
            print(names[i] + "'s device is NOT present")

arp_scan()
```

- Test this program and make sure it works by placing a known device in the arrays. Run the program from the command line as before:

```
pi@sensePi:~/presence $ python presence-detect.py
Frank's device is present
Someone Else's device is NOT present
```

arp scan with python

.

Next we'll use the SenseHat to output the results.

SenseHAT

In this section you will ouput the presence detection result using SenseHAT. The SenseHAT can be controlled using the SenseHat Python module.

- If you have not already, install the python module by opening a terminal window on your RPi and running the following commands:

```
sudo apt-get update
sudo apt-get install sense-hat
sudo reboot
```

- First, let's just output the `result` to the 8x8 LED display. Update your program by adding the following import statement at the top of the python program:

```
from sense_hat import SenseHat

sense = SenseHat()
```

We will use two arrays to keep track of the names and corresponding device MAC addresses. Add the following array declarations to your program **just before the `arp-scan()` function.

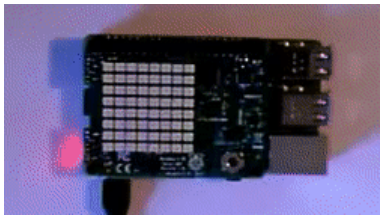
- Change the `arp_scan()` Python function to iterate through the `names` array and check the `arp-scan` for the corresponding MAC address. Replace the `arp_scan()` with the following code:

```
def arp_scan():
    output = subprocess.check_output("sudo arp-scan -l", shell=True)
    for i in range(len(names)):
        result = names[i]
        if macs[i] in output:
            result=result+" is home"
        else:
            result=result+" is not home"
        print(result)
        sense.show_message(result)
```

Now your program will, for each name:

- check the arp-scan `output` for the MAC address of his/her device.
- print the `result` to the console and show the result on the SenseHAT 8x8 LED.

Run the program as before and you should see something similar to the following:



sensehat 8x8 led

Automate Presence Detection

At the moment, you have to run your presence detection script manually on the RPi. Ideally, your script would run automatically at a set interval, perhaps every minute.

Loop forever

Add an endless loop in the program that waits of 60 seconds after every successful call to the apr-scan() function. You can use the `sleep()` function in the `time` python package to do this.

- In `presence-detector.py` , add the following import statement at the top of the file:

```
from time import sleep
```

- Put the call to the `arp_scan()` function in an endless loop(`while True:`) and add code to suspend execution for 60 seconds(`sleep(60)`) on each iteration:

```
while True:
    arp_scan()
    sleep(60)
```

- Run the program again. This time your program should never stop and should repeat the scan every 60 seconds.

When things go wrong...

Usually, when perfoming any input/output or network connections with programming, it is a good idea to assume that at some stage the connection will fail or that the file you are trying to use might become unavailable. You can think of this as an "exceptional" event - you're not expecting it but it might happen. Also, if we are running a device remotely and will not be attending, it may be a good idea to start logging information if an error or excpetional event does occur so that you can debug it.

In Python, error handling at run time in done through the use of exceptions that are caught in try blocks and handled in except blocks. We can then using logging to save details of the error for debugging later.

- Add the following logging declaration to the `precence-detector.py` program

```
import logging

logging.basicConfig(filename='presence-detector.log',level=logging.INFO, format='%(asctime)s - %(message)s')
logging.info('Starting presencenan detector')
```

Currently, in our `presence-detector.py` program, if an error occurs in the arp-scan() function the program will terminate.

- In the arp-scan() function, surround the code with a try-except block that logs any errors that occur:

```
def arp_scan():
    try:
        output = subprocess.check_output("sudo arp-scan -l", shell=True)
        for i in range(len(names)):
            result = names[i]
            if macs[i] in output:
                result=result+" is home"
            else:
                result=result+" is not home"
            print(result)
            sense.show_message(result)
    except Exception as e:
        logging.error(e)
```

- Now run your program as before. It should work exactly as before but now you will have a log file in the same directory as your script. Check it's contents and you should see similar to the following:

2018-10-16 09:09:11,225 - Starting presence detector

Crontab

cron is a utility that allows tasks to be automatically run in the background at regular intervals. The Crontab (CRON TABle) is a file which contains the schedule of cron entries to be run and at specified times. The crontab File location varies by operating system however you can easily access it on the RPi using the `crontab` utility program.
In a terminal window, enter `crontab -e` at the prompt:

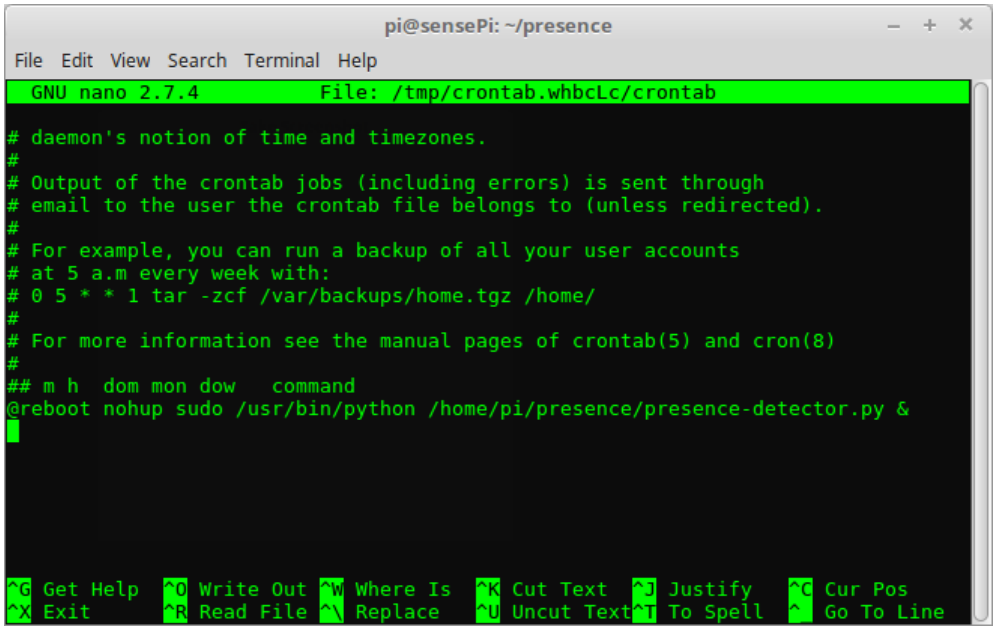
```
pi@sensePi:~/presence $ sudo crontab -e
no crontab for root - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/ed
  2. /bin/nano        <---- easiest
  3. /usr/bin/vim.tiny

Choose 1-3 [2]:
```

crontab

Follow the instructions and select your favourite editor (default is nano). At the end of the file, add `@reboot nohup sudo /usr/bin/python /home/pi/presence/presence-detector.py &` :



crontab

Save and exit the Crontab and reboot the RPi by typing `sudo reboot` at the command prompt. The presence detector should now start in the background on every reboot of the RPi.

- Check the log file now just to see if everything is OK. You may notice some errors the first time the script executes. If you do, try to come up with a reason and investigate a solution using the web.
- If you do not get errors, how would you know if the simple exception handling/logging works? Propose a way to force an error in the try block of the code.

Exercises/Report

Include the following in your lab report

1. If possible, include screen shots of your program working.
2. Lessons learned
3. Issues encountered and how you resolved them.
4. On the Sensehat, change the program to display the names of those who are present in green and not present in red
5. The current program requires you to "hardcode" names and MAC addresses into the program. Try to update the program so that names and MAC addresses are read from a file or files. (e.g. create a file called `devices.dat` and read the file when the program starts).
6. Final version of the code you created (or a link to an online repository)