Project: Scapy Wi-Fi Scanner

Ryan Possidel and Sultan Taj

NCS 350 Spring 2023

Abstract

This project takes a look at using Scapy to make a Wi-Fi scanner with a Kali laptop and a Panda wireless network interface card. This project is python code using Scapy to scan the Wi-Fi for networks on channel 1, 6 and 11. The scan will show us the BSSID, SSID, dBm signal strength and encryption method. This is great information to know if you want to know what networks are around you and information about those networks. The information that is output is information you would want to know if you wanted to understand the network better.

Introduction

In this project we present a Scapy based Wi-Fi scanner for network analysis. The scanner is able to detect the BSSID, SSID, channel number, signal strength and what security it is using. This program was written in Scapy which is a packet manipulation library written in python. We run this program on a Kali linux laptop using a Panda wireless network interface card as our WLAN1 interface. We use a script to be able to put the card into monitor mode and when we scan to be able to change the channel we are scanning on. The Wi-Fi scanner can be used for network troubleshooting, site surveying and a security assessment. In this report we will go over the motivation for the project, then the background information of the technology, then the steps and results of the project, then wrap everything up in the conclusion and some suggestions for further improvements to the project.

Motivation

There were several motivations for wanting to make the Wi-Fi scanner using Scapy. The first is that Scapy is a very useful and interesting tool to use and we wanted to get more experience with using it. Scapy can manipulate packets in many different ways and give you information of those packets that it scans. The second is that we wanted to create a Wi-Fi scanner because we wanted to see what other networks are around us that we don't know about. We also wanted to know more information about the networks around us and if needed to troubleshoot network problems we would have a scanner to help us figure it out. This is a very useful tool to learn in the world of cybersecurity, Scapy is highly customizable to suit your packet manipulation needs.

Background

Scapy is a powerful interactive packet manipulation library written in python. For this project we will be using Scapys's sniff function. This function sniffs for packets and returns information about the packets that were sniffed. The sniff function can be used in many different ways. You can choose what types of packets, how many packets, what interface, and it can save the sniffed packets to a pcap file. One very useful feature of sniff is the prn option, this allows you to pass a

function every time a packet is sniffed. Our code will output a few different pieces of information. The first two pieces of information the code will output is the BSSID and SSID. The BSSID is the mac address of the access point or router and the SSID is the name of the network it is on. The next piece of information is the channel number. Different networks are able to operate on the same hardware because of different channels. Networks are usually on channels 1, 6 or 11 because there is enough spacing between them so there is no channel overlapping. The next piece of information is the signal strength. This is useful because the strength can determine the distance the network is from you. If the network is weaker it should mean that the network is farther away from you. The last piece of information is the security of the network. It will output if it is using WEP, WPA, WPA2 etc. All of this information is important to learn and understand what networks are around you and what information is easily visible to an attacker.

Procedures and Results

1. In order to start this lab we are going to make sure that our Linux device, (Kali Laptop), has both Python and Scapy installed. To do so there are a few commands we can input in the command line. First we are going to install Python onto our device by inputting:

   #sudo apt-get install python

   After installing python, we are now able to obtain Scapy through the command line or python. The commands used are:

   #**sudo apt install scapy (Command Line)**
   >install scapy (Python Command Line)

   Note: The system used for this project was one of the Kali Laptops, which came preinstalled with Python 2.7 and Scapy 2.5.0

2. Now that we have Scapy installed along with Python, we can now start to write our Wi-Fi scanner. To do so we are going to create a python text file using a text editor of your choice, (I personally chose to use Nano), and provide it with executable permissions. The commands to perform this are show below:

   #sudo nano wifi_scan.py
   #sudo chmod 700 wifi_scan.py

   Note: The 700 within the chmod makes the file executable only to you and you alone

3. With our file now created we are going to write our code to detect local networks. When starting our code it is important to import the files used from scapy in order to have our scanner functional. Within the python text file, we are going to write:

   from scapy.all import (Dot11, Dot11Beacon, Dot11Elt)
   from scapy.sendrecv import sniff

   Note:

   To make sure these are functional within our program we can open up python through the command: **sudo python** , and import in these processes.

   >from scapy.all import (Dot11, Dot11Beacon, Dot11Elt)
   >from scapy.sendrecv import sniff

4. This portion of the code will be the function used to check if the broadcasted packets contain an 802.11 frame with a Beacon Frame.

   def scanner(packet):
   if packet.haslayer(Dot11Beacon) and if packet.haslayer(Dot11):

5. Within this function and if statement we are going to create the variables we are looking for and how we are going to obtain them. These are going to be the BSSID, SSID, dBm signal strength, channel number, and encryption type.

   bssid = packet[Dot11].addr3 or “ff:ff:ff:ff:ff:ff”
   ssid = packet[Dot11Elt].info.decode()
   dbm_signal = packet.dBm_AntSignal
   channel = int(ord(packet[Dot11Elt:3].info))
   encryption = packet[Dot11Beacon].network_stats().get(“encryption”, None)

   The BSSID information is being grabbed through Dot11.addr3 which is the broadcast address, to prevent any mixups we also added in the broadcast address manually.
   The SSID is received through the Dot11Elt frame.
   Note: This current SSID config will provide the SSID frame if public but if private, nothing will be shown as an output to the user.
   The signal strength is received from its corresponding packet
   The channel is received through Dot11Elt packet
   The type of encryption used on the network is obtained through the Beacon Frame in its network stats

6. Now that we have each of our variables assigned, we are going to print out our findings from these frames. We can do through a print command as seen below:

   print("BSSID: %s, SSID: %s, Channel: %d, Signal: %d, Encryption: %s" % (bssid, ssid, channel, dbm_signal, encryption))

   Note: This command is written to work in Python 2.7. For current versions of python you can write out the command in the way seen below:

   print(f"BSSID: {bssid}, SSID: {ssid}, Signal: {dbm_signal}, Channel: {channel}, Encryption: {encryption}")

7. Before we can print our final results, we need to assign the interface we are going to use for the scan. To do so we are first going to assign which interface we will be using for the program. To find your wireless interface you can input the command **iwconfig** into the command line to find it. Then within your code you create the line:

   interface = "wlan1"

   Note: The interface I used for this program was wlan1 but for you it may be different, it is important to see which interface you'll be using yourself.

8. With our interface chosen, we are going to write a print statement stating a scan has started, along with a sniff command. The sniff command is what's used to detect packets and in this case, print them as well. The commands can be seen below:

   print("Scanning for Wi-Fi networks on %s…" % interface)
   sniff(iface=interface, prn=scanner, count=10, timeout=30)

   Within our sniff command, we have a couple options shown. iface regards to the interface used so we assigned it to our interface variable but you can also use the interface name itself, prn prints out whatever you assign it to which in our case is our function scanner, count is the amount of times it will output a network which we set as 10, and timeout refers to how long the program will run before closing which is set to 30 seconds.

Shown below is the fully written program used in this project to create our Wi-Fi Scanner

Figure 1: Python code of Wi-Fi Scanner (Written in Python 2.7 and Scapy 2.5.0)

9. To have our code fully functional, we need to set our device into monitor mode. A seamless way to integrate this with our python code is to add it into a shell script. The commands we will need to create this script are: **sudo nano wifi_scan.sh** (to create the script), and **sudo chmod 700 wifi_scan.sh** (to make the script executable). Inside of the shell script file, we will have our python code along with the series of commands to set monitor mode and onto the three main channels for simplicity. It can be seen below:

#!/bin/bash

sudo ifconfig wlan1 down
sudo iwconfig wlan1 mode monitor
sudo iwconfig wlan1 ch1
sudo rfkill unblock all
sudo ifconfig wlan1 up

sudo python wifi_scan.py

Note: To have this code switch between channels, I simply repeated the commands seen above with channel 1, 6, and 11. This way after running the program it then switches to another channel and runs it again.

Shown below is the full script used to activate monitor mode, switch between channels and run our python program all in the script.

```
  GNU nano 4.3                                                  wifi_scan.sh
#!/bin/bash

sudo ifconfig wlan1 down
sudo iwconfig wlan1 mode monitor
sudo iwconfig wlan1 ch 1
sudo rfkill unblock all
sudo ifconfig wlan1 up

sudo python wifi_scan.py

sudo ifconfig wlan1 down
sudo iwconfig wlan1 mode monitor
sudo iwconfig wlan1 ch 6
sudo rfkill unblock all
sudo ifconfig wlan1 up

sudo python wifi_scan.py

sudo ifconfig wlan1 down
sudo iwconfig wlan1 mode monitor
sudo iwconfig wlan1 ch 11
sudo rfkill unblock all
sudo ifconfig wlan1 up
```

Figure 2: Shell script written for our Wi-Fi scanner to be functional

10. The final thing left to do now is to run the code and see if it outputs. We can do this by running our shell script through the command line with **sudo ./wifi_scan.sh** . When running this command, the output should look the screenshot below:

```
netlab@NCSKali-06:~$ sudo ./wifi_scan.sh
Scanning for Wi-Fi networks on wlan1...
BSSID: 8e:15:44:aa:5d:15, SSID: Employee, Channel: 1, Signal: -74, Crypto: set(['WPA2'])
BSSID: 88:15:44:aa:5d:15, SSID: Student, Channel: 1, Signal: -88, Crypto: set(['WPA2'])
BSSID: 86:15:44:aa:5d:15, SSID: eduroam, Channel: 1, Signal: -92, Crypto: set(['WPA2'])
BSSID: b6:15:44:aa:5d:15, SSID: , Channel: 1, Signal: -88, Crypto: set(['WPA2'])
Scanning for Wi-Fi networks on wlan1...
BSSID: 8e:15:44:aa:5d:31, SSID: Employee, Channel: 6, Signal: -64, Crypto: set(['WPA2'])
BSSID: 82:15:44:aa:5d:31, SSID: Visitor, Channel: 6, Signal: -66, Crypto: set(['OPN'])
BSSID: 86:15:44:aa:5d:31, SSID: eduroam, Channel: 6, Signal: -68, Crypto: set(['WPA2'])
BSSID: b6:15:44:aa:5d:31, SSID: , Channel: 6, Signal: -66, Crypto: set(['WPA2'])
Scanning for Wi-Fi networks on wlan1...
BSSID: 88:15:44:aa:5d:2a, SSID: Student, Channel: 11, Signal: -70, Crypto: set(['WPA2'])
BSSID: 8e:15:44:aa:5d:2a, SSID: Employee, Channel: 11, Signal: -70, Crypto: set(['WPA2'])
BSSID: 82:15:44:aa:5d:2a, SSID: Visitor, Channel: 11, Signal: -70, Crypto: set(['OPN'])
BSSID: 86:15:44:aa:5d:2a, SSID: eduroam, Channel: 11, Signal: -72, Crypto: set(['WPA2'])
BSSID: b6:15:44:aa:5d:2a, SSID: , Channel: 11, Signal: -68, Crypto: set(['WPA2'])
BSSID: 88:15:44:aa:5d:29, SSID: Student, Channel: 11, Signal: -70, Crypto: set(['WPA2'])
BSSID: 8e:15:44:aa:5d:29, SSID: Employee, Channel: 11, Signal: -68, Crypto: set(['WPA2'])
BSSID: 82:15:44:aa:5d:29, SSID: Visitor, Channel: 11, Signal: -68, Crypto: set(['OPN'])
BSSID: 86:15:44:aa:5d:29, SSID: eduroam, Channel: 11, Signal: -68, Crypto: set(['WPA2'])
BSSID: b6:15:44:aa:5d:29, SSID: , Channel: 11, Signal: -68, Crypto: set(['WPA2'])
netlab@NCSKali-06:~$
```

Figure 3: Output of Wi-Fi Scanner

Looking at this output we can see that there are several networks shown such as the Student, Visitor, and Employee network here on campus among the three channels being scanned. We can see that each channel shares their own BSSID, the encryption type for every network is WPA2, except for Visitor which is Open, and the signal strength between each of the channels differ from each other.

Conclusion

A Scapy Wi-Fi scanner is a very powerful tool you can use to help troubleshoot your network, see and gather information about the network around you and help secure your network better. Scapy is a very powerful packet manipulation tool that is highly customizable to suit your needs. The program searches for networks on channel 1, 6 and 11 and outputs BSSID, SSID, channel, signal output and security which all can be used to understand and learn more about the networks around you. Through this program, we can integrate other tools such as airodump-ng to find hidden SSID, Wireshark to analyze packets within the networks shown, or even aircrack-ng to crack through networks shown in the scanner.

Bibliography

"Documentation," *Aircrack*. [Online]. Available: https://www.aircrack-ng.org/. [Accessed: 24-Apr-2023].

Scapy. [Online]. Available: https://scapy.net/. [Accessed: 23-Apr-2023].