Creating a Packet Sniffer

Ayesha Khan

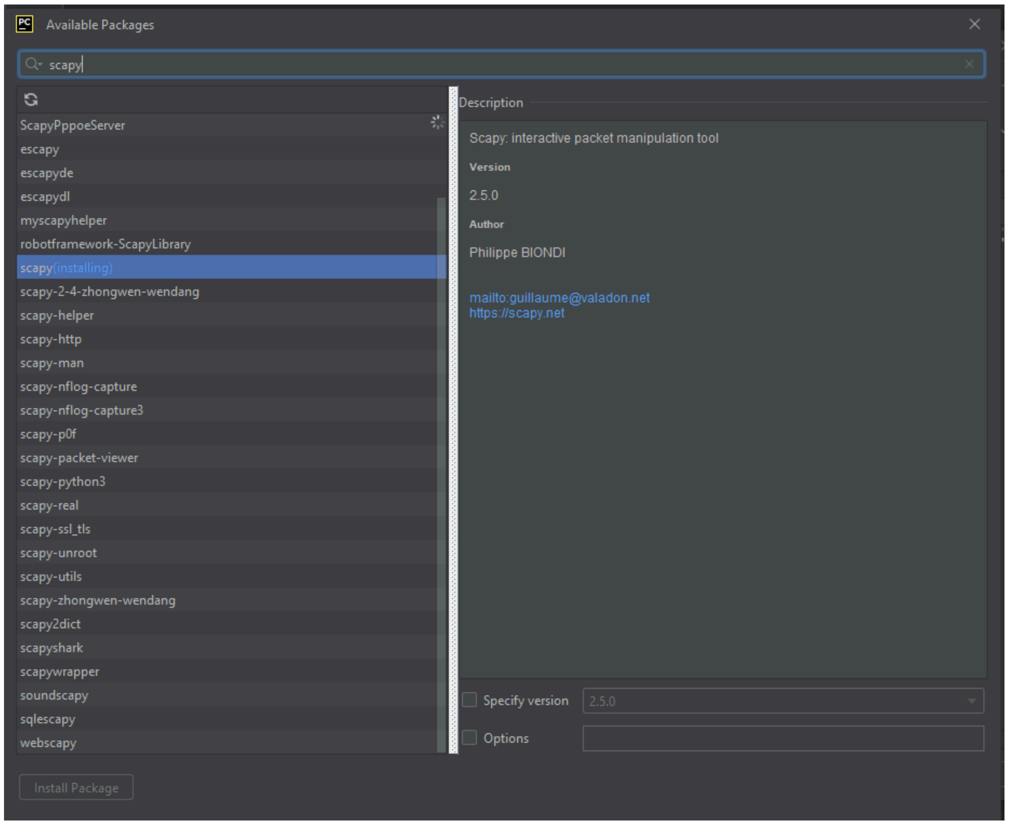
Things to Consider Before Getting Started:

1) **Understanding Networking Basics:** Ensure you have a solid understanding of how computer networks operate, including TCP/IP, OSI Model, and protocols like Ethernet, IP, TCP, and UDP.
2) **Choose a Programming Language:** Packet sniffers are often implemented in languages like Python, C/C++, or Java. Python is a good choice due to its readability and extensive libraries for networking.
3) **Learn Socket Programming:** You'll need to understand socket programming to capture and analyze network packets. Python's 'socket' module provides the necessary functionality.
4) **Capture Packets:** Use your chosen programming language to create a socket and capture packets from the network interface. You may need to use libraries like libpcap (in C/C++) or Scapy (in Python) to make this process easier.
5) **Analyze Packets:** Once you've captured packets, you can analyze them to extract useful information. This might include parsing headers, extracting payloads, and interpreting packet contents.
6) **Implement Filters and Rules:** Add functionality to filter packets based on criteria like source/destination IP, protocol type, port numbers, etc.
7) **Display Results:** Decide how you want to present the captured packet information to the user. This could be as simple as printing to the console or creating a graphical interface.
8) **Handle Errors and Excepting:** Make sure your program gracefully handles errors and exceptions, such as network errors or invalid packet formats.
9) **Testing and Debugging:** Thoroughly test your packet sniffer to ensure it works correctly in various network environments and conditions. Use debugging tools to troubleshoot any issues you encounter.
10) **Consider Security and Ethics:** Packet sniffers can be powerful tools, but they can also be used maliciously. Be sure to use your sniffer responsibly and ethically, and be aware of legal and privacy implications.

Getting Started:

Python will be the language used in this project.

Start by installing the scapy library on Python 3 since it will be a library we will be importing for this project.



Description

Scapy: interactive packet manipulation tool

Version

2.5.0

Author

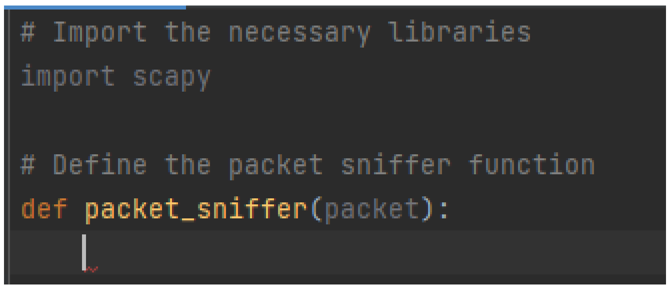
Philippe BIONDI

mailto:guillaume@valadon.net
https://scapy.net

Specify version 2.5.0

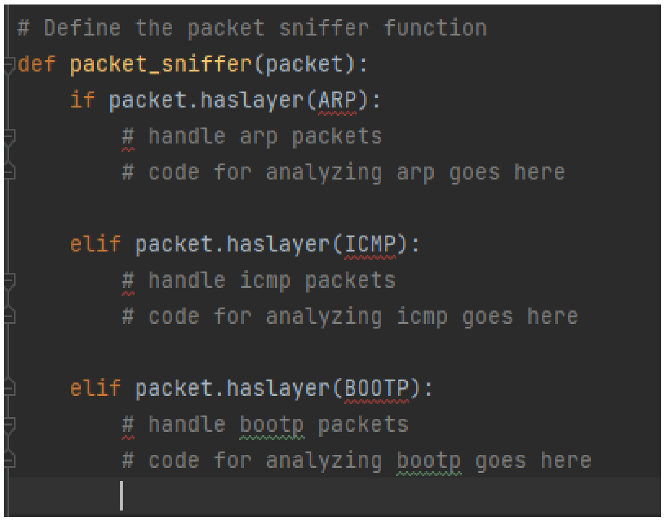
Options

Then we import the necessary libraries needed for the packet sniffer. After doing so, we start the code off by defining the packet sniffer function.



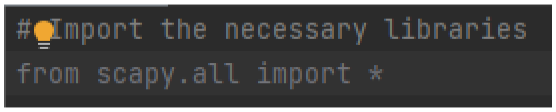
```
# Import the necessary libraries
import scapy

# Define the packet sniffer function
def packet_sniffer(packet):
    
```

Right here is a little bit of setup for the code.



```
# Define the packet sniffer function
def packet_sniffer(packet):
    if packet.haslayer(ARP):
        # handle arp packets
        # code for analyzing arp goes here

    elif packet.haslayer(ICMP):
        # handle icmp packets
        # code for analyzing icmp goes here

    elif packet.haslayer(BOOTP):
        # handle bootp packets
        # code for analyzing bootp goes here
        
```

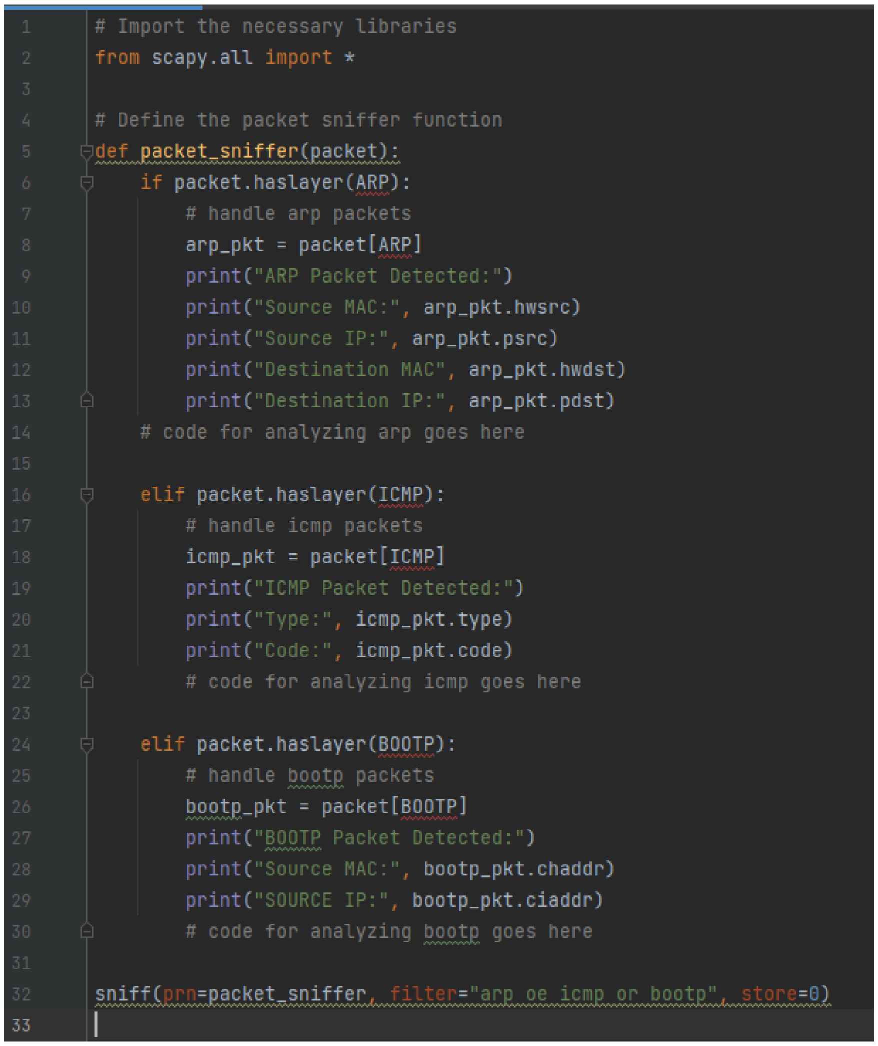
After doing some more research, I have decided to change importing scapy to importing scapy.all to ensure that all the required components are imported.



```
#Import the necessary libraries
from scapy.all import *
```

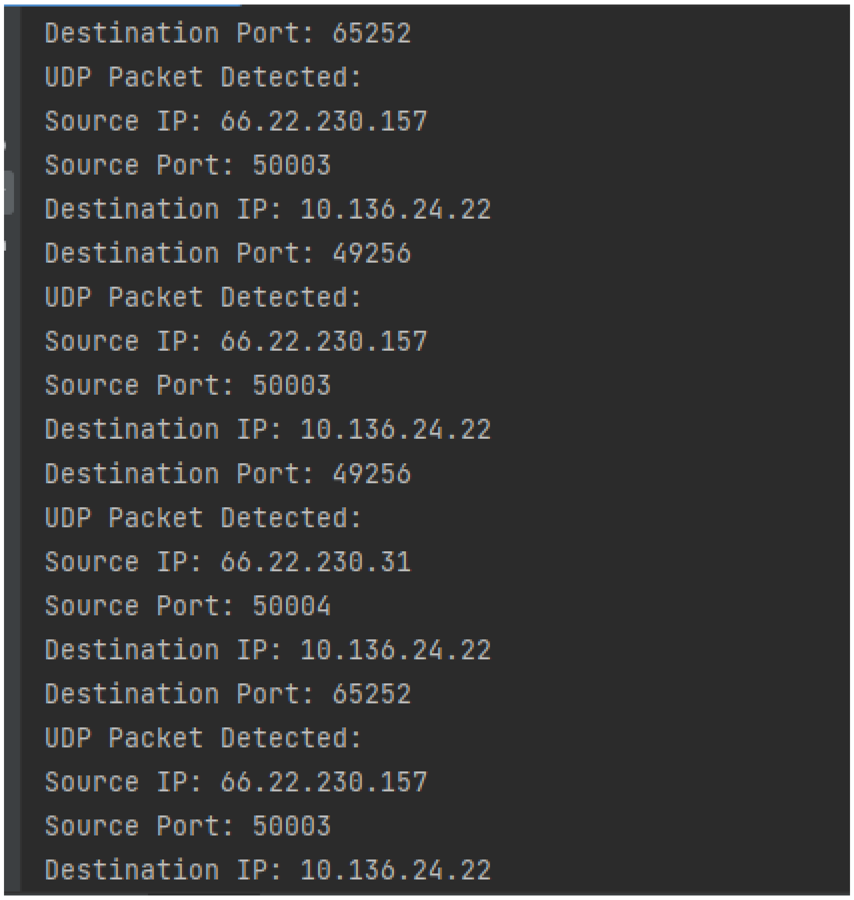
After changing the library import, we move onto the rest of the code of the packet sniffer. Here are the bullet points:

- For each packet type (ARP, ICMP, BOOTP), we've accessed relevant fields using '*packet[layer]*'.
- We've printed out some basic information about each packet type for demonstration purposes, but we can later modify the code to perform more complex analysis depending on our requirements.
- Finally, we've used the *'sniff'* function from Scapy to start capturing packets, passing *'packet_sniffer'* as the callback function. The *'filter'* parameter allows filtering packets based on specific protocols.



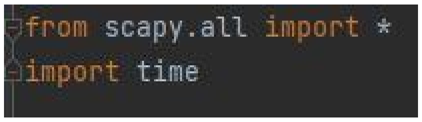
```
# Import the necessary libraries
from scapy.all import *

# Define the packet sniffer function
def packet_sniffer(packet):
    if packet.haslayer(ARP):
        # handle arp packets
        arp_pkt = packet[ARP]
        print("ARP Packet Detected:")
        print("Source MAC:", arp_pkt.hwsrc)
        print("Source IP:", arp_pkt.psrc)
        print("Destination MAC", arp_pkt.hwdst)
        print("Destination IP:", arp_pkt.pdst)
    # code for analyzing arp goes here

    elif packet.haslayer(ICMP):
        # handle icmp packets
        icmp_pkt = packet[ICMP]
        print("ICMP Packet Detected:")
        print("Type:", icmp_pkt.type)
        print("Code:", icmp_pkt.code)
        # code for analyzing icmp goes here

    elif packet.haslayer(BOOTP):
        # handle bootp packets
        bootp_pkt = packet[BOOTP]
        print("BOOTP Packet Detected:")
        print("Source MAC:", bootp_pkt.chaddr)
        print("SOURCE IP:", bootp_pkt.ciaddr)
        # code for analyzing bootp goes here

sniff(prn=packet_sniffer, filter="arp oe icmp or bootp", store=0)
```

I did some more research and decided to add in some more protocols to the code. So we're going to be adding TCP and UDP, two very common protocols you see in pcaps.

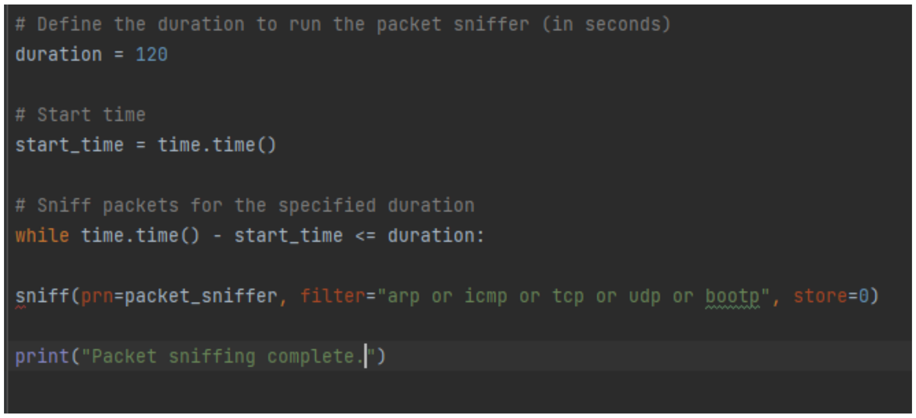


```
Destination Port: 65252
UDP Packet Detected:
Source IP: 66.22.230.157
Source Port: 50003
Destination IP: 10.136.24.22
Destination Port: 49256
UDP Packet Detected:
Source IP: 66.22.230.157
Source Port: 50003
Destination IP: 10.136.24.22
Destination Port: 49256
UDP Packet Detected:
Source IP: 66.22.230.31
Source Port: 50004
Destination IP: 10.136.24.22
Destination Port: 65252
UDP Packet Detected:
Source IP: 66.22.230.157
Source Port: 50003
Destination IP: 10.136.24.22
```

After adding in the two protocols and running the packet sniffer, thankfully, it works! However, it tends to run until I cancel it myself. So I realize I need to put a timer on how long I want it to run for. So I look up how to add a timer into the code so we can have it run for a set amount of time.
We import the time library, and then add in the code for the timer.

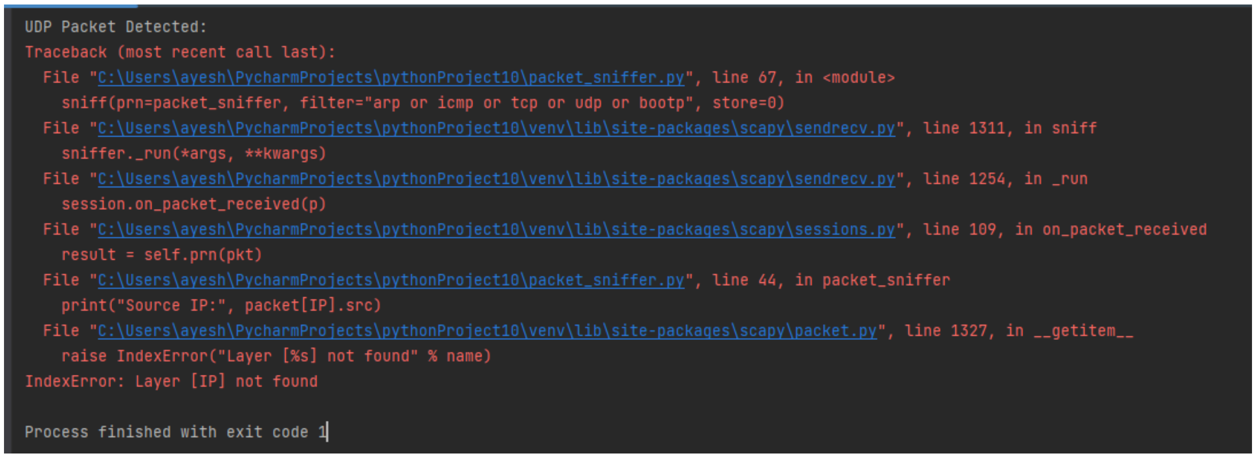


```python
from scapy.all import *
import time
```



```python
# Define the duration to run the packet sniffer (in seconds)
duration = 120

# Start time
start_time = time.time()

# Sniff packets for the specified duration
while time.time() - start_time <= duration:

sniff(prn=packet_sniffer, filter="arp or icmp or tcp or udp or bootp", store=0)

print("Packet sniffing complete.")
```

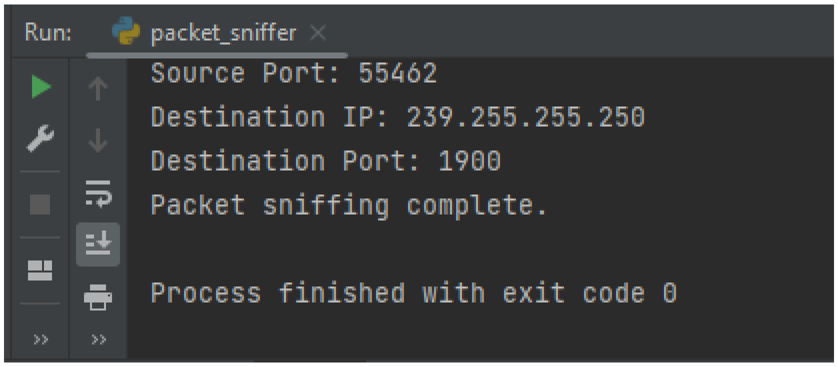
I encountered a lot of errors when I tried to use the packet sniffer after adding the timer. I tried a couple of things to help with the errors, like I fixed the indent to the *sniff* line, and I also imported more libraries. After doing so I ran it again and it seemed to run for a couple of seconds before getting an error again, so I am a little stuck now.



```
UDP Packet Detected:
Traceback (most recent call last):
  File "C:\Users\ayesh\PycharmProjects\pythonProject10\packet_sniffer.py", line 67, in <module>
    sniff(prn=packet_sniffer, filter="arp or icmp or tcp or udp or bootp", store=0)
  File "C:\Users\ayesh\PycharmProjects\pythonProject10\venv\lib\site-packages\scapy\sendrecv.py", line 1311, in sniff
    sniffer._run(*args, **kwargs)
  File "C:\Users\ayesh\PycharmProjects\pythonProject10\venv\lib\site-packages\scapy\sendrecv.py", line 1254, in _run
    session.on_packet_received(p)
  File "C:\Users\ayesh\PycharmProjects\pythonProject10\venv\lib\site-packages\scapy\sessions.py", line 109, in on_packet_received
    result = self.prn(pkt)
  File "C:\Users\ayesh\PycharmProjects\pythonProject10\packet_sniffer.py", line 44, in packet_sniffer
    print("Source IP:", packet[IP].src)
  File "C:\Users\ayesh\PycharmProjects\pythonProject10\venv\lib\site-packages\scapy\packet.py", line 1327, in __getitem__
    raise IndexError("Layer [%s] not found" % name)
IndexError: Layer [IP] not found

Process finished with exit code 1
```

After doing some more research I came across trying to use the *getlayer* instead of just *packet.layer.* So let's try that.

Now for these errors, I do not have screenshots for because I forgot to take some while working on them. I got extremely focused on trying to get the code to run and fix the errors that it slipped my mind.
I first tried importing some more libraries to see if that was the issue. It resolved some of the errors, however, it still would not run. To the libraries I added *from scapy.all import sniff* to make the sniff function work. That resolved the issue for 'sniff' however there were still some more errors revolving around IP. I then added *import scapy.layers.inet import IP,* and that seemed to fix the problem.
When I ran the code again, it seemed to work, however, it would keep running and not implement the timer function I had added. So I had to manually stop the packet sniffer from running and when I did that, it continuously kept printing the 'Packet sniffing complete.' message constantly until I stopped it myself. To fix that, I added *timeout=duration* to the sniff function so it could stop running after 2 minutes. After doing so, the packet sniffer runs perfectly!



```
Source Port: 55462
Destination IP: 239.255.255.250
Destination Port: 1900
Packet sniffing complete.

Process finished with exit code 0
```