Mini Project for Digital Forensic Workshop

Problem Statement: Capture network packets in a PCAP file using Scapy in a Python environment. Develop Python code to identify anomalies in the given packets inside the PCAP file based on the following rules:

- **Rule 1**: Common destination ports for TCP and UDP.
- **Rule 2:** Excessive Traffic (DDoS).
- **Rule 3:** Number of packets and packet size.
- **Rule 4:** Unsolicited ARP replies.
- **Rule 5:** Unusually large DNS responses.
- **Rule 6:** Excessive ICMP Echo requests.
- **Rule 7:** Excessive TCP SYN.
- **Rule 8:** IPs scans excessive ports.

The code ultimately creates a report.csv file that will contain the following columns:

IP Address	MAC Address	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	MDP(%)
ip_1	mac_1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
ip_2	mac_2	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
-	-	-	-	-	-	-	-	-	-	
ip_n	mac_n	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	

If a device violates any rule, the corresponding cell value will be 1; otherwise, it will be 0. The last column calculates the **Malicious Device Probability (MDP)** as a percentage, which can be defined as:

MDP(%)=(Rule1+Rule2+Rule3+Rule4+Rule5+Rule6+Rule7+Rule8)X100/8

Example:

```
from scapy.all import rdpcap, DNS, IP, ICMP, TCP, ARP
from collections import Counter
from collections import defaultdict
import time

# Load the PCAP file
```

```
packets = rdpcap('example.pcap')

# Inspect packets
for packet in packets:
    print(packet.summary())

non_standard_ports = set()
################################################################
# Rule 1: Detecting Traffic on Non-Standard Ports
################################################################
for packet in packets:
    if packet.haslayer('TCP'):
        tcp_layer = packet['TCP']
        if tcp_layer.dport not in [80, 443, 22]:  # Add standard
destination ports
            non_standard_ports.add(tcp_layer.dport)

print("Non-standard ports detected:", non_standard_ports)
################################################################
# Rule 2: High Traffic Volume (DDoS Detection)
################################################################
ip_count = Counter()

#Advantages of Using Counter:Easy Initialization: No need to
predefine keys or set values #manually, Automatic Counting: When
an element is not in the Counter, it is initialized with a
#count of 0,

#Built-in Operations: most_common(n): Returns the n most common
elements.Supports #arithmetic operations #between Counters.


for packet in packets:
    if packet.haslayer('IP'):
        ip_layer = packet['IP']
        ip_count[ip_layer.src] += 1
################################################################
# Rule 3: Detect IPs exceeding a threshold
################################################################
threshold = 100  # Set your threshold
MAX_MTU=1500#MTU (Maximum Transmission Unit) is the largest size of a data
packet that can be transmitted over a network or communication protocol
without needing to be fragmented.
ddos_candidates = [ip for ip, count in ip_count.items() if count >
threshold]
print("Potential DDoS IPs:", ddos_candidates)

#Packet Size Analysis
```

```
for packet in packets:
    size = len(packet)
    if size > MAX_MTU:  # MTU size exceeds standard Ethernet
        print(f"Large packet detected: {size} bytes")

################################################################
#Rule 7: Detect TCP SYN Flood (High number of SYN packets)
################################################################
SYN_FLOOD_THRESHOLD = 100  # Number of SYN packets in a short period
syn_count = defaultdict(int)
for packet in packets:
    if packet.haslayer(TCP) and packet['TCP'].flags == 0x02:  # SYN flag
set
        src_ip = packet[IP].src
        syn_count[src_ip] += 1

for ip, count in syn_count.items():
    if count > SYN_FLOOD_THRESHOLD:
        print(f"Potential TCP SYN Flood from {ip}, {count} SYN packets")
################################################################
################################################################
# Rule 8:Port Scanning Detection
################################################################
PORT_SCAN_THRESHOLD = 5  # Connection attempts on multiple ports from the
same IP
connection_attempts = defaultdict(set)  # Source IP -> Set of destination
ports
for packet in packets:
    if packet.haslayer(TCP):
        tcp_layer = packet['TCP']
        if packet.haslayer(IP):
            connection_attempts[packet[IP].src].add(tcp_layer.dport)

for ip, ports in connection_attempts.items():
    if len(ports) > PORT_SCAN_THRESHOLD:
        print(f"Potential Port Scan detected from {ip} targeting
{len(ports)} ports")
```