01 Using the Python library Scapy, analyze the network packets associated with the suspicious IP address provided.

1. Packet Capture: Scapy can capture packets from the network interface in real-time.
2. Packet Dissection: Scapy can dissect packets and extract useful information such as source and destination IP addresses, ports, protocols, and payload.
3. Packet Filtering: Scapy allows filtering of packets based on specific criteria (e.g., IP address, protocol) to focus on relevant traffic.
4. Packet Crafting: Scapy can forge custom packets for testing and simulation purposes.

1. Install Scapy

To install the scapy we can go through this steps and if already installed we can check it by the following methods. We can access the scapy with the interactive terminal of python or by a python file and import the scapy and its sniffing function “sniff”.

| pip install scapy#For systems that require administrator privileges, you may need to use sudo (Linux/macOS):sudo pip install scapy#Or, if you're using a virtual environment (which is recommended for Python projects):python -m venv myenvsource myenv/bin/activate # On Windows use `myenv\Scripts\activate`pip install scapy# Check the Installed VersionpythonCopy codeimport scapyprint(scapy.\_\_version\_\_)Open a Python interpreter by typing python or python3 in your terminal or command prompt. |
| --- |

2. Capture Network Traffic and Store in a File:

To capture network traffic, we need to specify the network interface and set up a packet sniffer.

| from scapy.all import sniff, wrpcap, IP# Define the network interfaceinterface = "eth0"# Define the file to store captured packetspcap\_file = "captured\_traffic.pcap"# Define a packet capture functiondef packet\_capture(packet): # Append packets to the file wrpcap(pcap\_file, packet, append=True) print(f"Packet captured: {packet.summary()}")# Start packet capture (this will run indefinitely, stop manually)sniff(iface=interface, prn=packet\_capture, store=0) |
| --- |

3. Filter Packets by IP Address

To focus on traffic originating from a particular IP address, we want to use for filter

| from scapy.all import sniff# Define the network interfaceinterface = "eth0"# Define the suspicious IP addresssuspicious\_ip = "192.168.1.100"# Define a packet capture functiondef packet\_capture(packet): if packet.haslayer(IP): if packet[IP].src == suspicious\_ip or packet[IP].dst == suspicious\_ip: print(packet.summary())# Start packet capturesniff(iface=interface, prn=packet\_capture, store=0) |
| --- |

Step 4: Analyze Captured Packets

To analyze the captured packets in more detail.

| from scapy.all import sniff, IP, TCP, UDP# Define the network interfaceinterface = "eth0"# Define the suspicious IP addresssuspicious\_ip = "192.168.1.100"# Define a packet capture functiondef packet\_capture(packet): if packet.haslayer(IP): if packet[IP].src == suspicious\_ip or packet[IP].dst == suspicious\_ip: print(f"Packet: {packet.summary()}") print(f"Source IP: {packet[IP].src}") print(f"Destination IP: {packet[IP].dst}") if packet.haslayer(TCP): print(f"Source Port: {packet[TCP].sport}") print(f"Destination Port: {packet[TCP].dport}") print(f"TCP Flags: {packet[TCP].flags}") elif packet.haslayer(UDP): print(f"Source Port: {packet[UDP].sport}") print(f"Destination Port: {packet[UDP].dport}")# Start packet capturesniff(iface=interface, prn=packet\_capture, store=0) |
| --- |

Identification and Interpretation of Suspicious or Anomalous Network Behavior

Suspicious Indicators:

1. Unusual Port Activity: Traffic on uncommon ports might indicate malicious activity.
2. High Traffic Volume: A high volume of traffic from a single IP can suggest a DDoS attack or data exfiltration.
3. Irregular Protocol Usage: Use of unusual protocols might indicate an attempt to bypass network defenses.
4. Strange Payloads: Payloads containing suspicious data patterns or commands could indicate malware communication.

Expected Code: 1. Write a python code to Network Packet Analysis with Scapy

| from scapy.all import sniff, wrpcap, IP# Define the suspicious IP addresssuspicious\_ip = "192.168.1.100" # Replace with the actual suspicious IP address# Function to filter and analyze packetsdef packet\_callback(packet): if packet.haslayer(IP) and (packet[IP].src == suspicious\_ip or packet[IP].dst == suspicious\_ip): print(packet.summary()) packet.show()# Capture packets and apply the callback functionpackets = sniff(prn=packet\_callback, count=100)# Save captured packets to a pcap file for further analysissuspicious\_packets = [pkt for pkt in packets if pkt.haslayer(IP) and (pkt[IP].src == suspicious\_ip or pkt[IP].dst == suspicious\_ip)]wrpcap("suspicious\_packets.pcap", suspicious\_packets) |
| --- |

**Question - 2**

**Steps to Develop the Detection System**

1. Extract Features: Analyze various characteristics of the website URLs.
2. Prepare a Dataset: For training a model, you would typically use a dataset containing both phishing and legitimate websites.
3. Train a Model: Use a machine learning model to classify URLs based on extracted features.
4. Predict Phishing: Apply the trained model to new URLs to determine if they are phishing sites.

**Prerequisites**

Ensure that we have the necessary libraries installed. You can install them using pip:

| pip install scikit-learn pandas requests beautifulsoup4 |
| --- |

**Dependencies**

1. numpy:
	* Purpose: Provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
	* Installation: pip install numpy
2. pandas:
	* Purpose: Offers data structures and data analysis tools, particularly useful for handling and analyzing data in tabular form.
	* Installation: pip install pandas
3. scikit-learn:
	* Purpose: A machine learning library that provides simple and efficient tools for data mining and data analysis. It includes various algorithms for classification, regression, clustering, and dimensionality reduction.
	* Installation: pip install scikit-learn
4. beautifulsoup4:
	* Purpose: Used for web scraping purposes to parse HTML and XML documents. Though it's not used directly in the provided script, it is useful for more complex implementations that involve scraping website content.
	* Installation: pip install beautifulsoup4
5. requests:
	* Purpose: Allows you to send HTTP requests easily. While it's not used in the basic script, it's useful for fetching web pages to analyze content, which could be added in more advanced versions.
	* Installation: pip install requests

Ensure Dependencies Are Installed: Make sure you have all necessary Python libraries installed.

**Prepare the Script:** Copy the entire script provided earlier into a Python file, say phishing\_detection.py.

**Run the Script:** Execute the script from the command line or a Python environment. The script includes predefined URLs for testing. You can modify these URLs or add new ones to test other sites.

**Test URLs:** To test additional URLs, update the real\_url and phishing\_url variables in the script with the URLs you want to check. You can also adapt the script to take URLs as input.

| import refrom urllib.parse import urlparseimport requestsdef extract\_features(url): features = {} # URL length features['url\_length'] = len(url) # HTTPS check features['https'] = url.startswith('https') # Check for the presence of suspicious keywords suspicious\_keywords = ['login', 'secure', 'update', 'account'] features['suspicious\_keywords'] = any(keyword in url for keyword in suspicious\_keywords) # Domain features parsed\_url = urlparse(url) domain = parsed\_url.netloc features['domain\_length'] = len(domain) features['subdomain\_count'] = len(domain.split('.')) - 1 features['has\_digits'] = bool(re.search(r'\d', domain))  return featuresdef analyze\_features(features): # Basic heuristics for phishing score = 0 if features['url\_length'] > 100: score += 1 if not features['https']: score += 1 if features['suspicious\_keywords']: score += 1 if features['subdomain\_count'] > 2: score += 1 if features['has\_digits']: score += 1 # Threshold for phishing detection return score >= 3def fetch\_url\_content(url): try: response = requests.get(url, timeout=5) return response.text except requests.RequestException: return ''def is\_phishing(url): features = extract\_features(url) # Analyze extracted features return analyze\_features(features)# Example usagereal\_url = 'https://www.example.com'phishing\_url = 'http://secure-login.example.com'print(f"Real URL is phishing: {is\_phishing(real\_url)}")print(f"Phishing URL is phishing: {is\_phishing(phishing\_url)}") |
| --- |