Imagine you are a cybersecurity analyst working for a large multinational corporation. One morning, your team receives an urgent report about a potential security breach in the company's network. The IT department has noticed unusual network activity originating from a particular IP address. Your team has been tasked with investigating this incident to determine if it poses a threat to the organization's network security. Assignment

Question:

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

Expected Procedure:

1. A detailed explanation of how Scapy can be utilized to capture and dissect network packets.
2. A step-by-step breakdown of the process you followed to capture and analyze the network traffic.
3. Identification and interpretation of any suspicious or anomalous network behavior observed in the captured packets.
4. Recommendations for mitigating the identified security risks and securing the network against similar threats in the future.

Expected Code:

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

Answer:

1. Explanation of Scapy:

   Scapy is a powerful Python library used for packet manipulation and network analysis. It allows cybersecurity analysts to perform tasks such as packet sniffing, packet crafting, network scanning, and protocol dissection. With Scapy, analysts can capture packets from the network, inspect their contents, and analyze network traffic for signs of suspicious or malicious activity.

2. Step-by-step Process:

a. Install Scapy:

- Ensure that Scapy is installed in your Python environment. You can install it using pip:

```
pip install scapy
```

b. Launch Python:

- Open a Python environment or script where you'll utilize Scapy for analysis.

c. Import Scapy:

- Import Scapy into your Python script:

```
from scapy.all import *
```

d. Capture Packets:

- Use Scapy's sniff function to capture packets. Specify the suspicious IP address in the filter:

```
captured_packets = sniff(filter="host <suspicious_IP>", count=<number_of_packets_to_capture>)
```

e. Analyze Packets:

- Analyze the captured packets to identify any suspicious behavior:

```
for packet in captured_packets:
# Analyze packet fields, headers, and payload
print(packet.summary())
```

f. Interpretation:

- Look for any anomalies or suspicious patterns in the captured packets. This may include:

- Unusual source or destination IPs

- Uncommon protocols or port numbers

- Large amounts of data transfer

- Attempts to establish unauthorized connections

3. Identification of Suspicious Behavior:

- Identify any packets that deviate from normal network traffic patterns. Look for signs of potential threats such as:

- Port scanning or reconnaissance activities

- Attempts to exploit known vulnerabilities

- Communication with known malicious IPs or domains

- Unencrypted sensitive data transmission

4. Recommendations for Mitigation:

- Implement robust network security measures such as firewalls, intrusion detection/prevention systems, and antivirus solutions.

- Conduct regular vulnerability assessments and patch management to address security weaknesses in the network infrastructure.

- Enforce strong access controls and authentication mechanisms to prevent unauthorized access to critical systems and data.

- Educate employees about cybersecurity best practices, including phishing awareness and safe browsing habits.

- Monitor network traffic continuously using tools like Scapy and SIEM (Security Information and Event Management) solutions to detect and respond to security incidents promptly.

- Establish incident response procedures to handle security breaches effectively and minimize their impact on the organization.

- Collaborate with other cybersecurity professionals and share threat intelligence to stay informed about emerging threats and security trends.

Code:

```
from scapy.all import *


def analyze_packets(suspicious_ip, packet_count):
    # Capture packets
    captured_packets = sniff(filter=f"host {suspicious_ip}", count=packet_count)


    # Analyze packets
    for packet in captured_packets:
        print("Source IP:", packet[IP].src)
        print("Destination IP:", packet[IP].dst)
        print("Protocol:", packet[IP].proto)
        print("Packet Summary:", packet.summary())
        print("")


if __name__ == "__main__":
    suspicious_ip = "192.168.1.100"  # Replace with the suspicious IP address
    packet_count = 10  # Number of packets to capture


    print(f"Analyzing network packets associated with {suspicious_ip}...")
    analyze_packets(suspicious_ip, packet_count)
```

Imagine you are working as a cybersecurity analyst at a prestigious firm. Recently, your company has been experiencing a surge in cyber attacks, particularly through phishing emails and websites. These attacks have not only compromised sensitive information but also tarnished the reputation of the company. In light of these events, your team has been tasked with developing a robust solution to detect and mitigate phishing websites effectively. Leveraging your expertise in Python programming and cybersecurity, your goal is to create a program that can accurately identify phishing websites based on various features and indicators.

Assignment Task:

Using the Python programming language, develop a phishing website detection system that analyzes website characteristics and determines the likelihood of it being a phishing site.

Expected Procedure:

1. Accept 2 web URL. One real and another one phishing.
2. Analyze the data from both the websites.
3. Identify the phishing site. Expected

Code:

1. Phishing Website Detection with Python

Answer:

To develop a phishing website detection system in Python, we can use machine learning techniques to analyze website characteristics and determine the likelihood of a website being a phishing site. Below is a basic outline of the steps involved:

1. Data Collection : Gather a dataset of labeled URLs, where each URL is labeled as either legitimate or phishing.

2. Feature Extraction : Extract relevant features from the URLs and associated web pages. Features may include:

- URL length
- Presence of HTTPS

- Domain age
- Presence of suspicious keywords in the URL or webpage content
- Number of external links
- Use of URL shorteners

3. Data Preprocessing : Prepare the dataset by encoding categorical features, handling missing values, and scaling numerical features if necessary.

4. Model Training : Train a machine learning model on the prepared dataset. Popular algorithms for binary classification tasks like this include Random Forest, Decision Trees, and Gradient Boosting classifiers.

5. Model Evaluation : Evaluate the trained model's performance using metrics such as accuracy, precision, recall, and F1-score on a separate validation dataset.

6. Website Analysis and Prediction : Given a new URL, extract its features and use the trained model to predict whether it is a phishing website or not.

Code:

```
 python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urlparse
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Function to extract features from a URL
def extract_features(url):
    parsed_url = urlparse(url)

    # Feature 1: URL length
    url_length = len(url)

    # Feature 2: Presence of HTTPS
    https_presence = 1 if parsed_url.scheme == "https" else 0

    # Feature 3: Domain age (in days)
    try:
        creation_date = datetime.strptime(parsed_url.netloc.split(".")[-1], "%Y-%m-%d")
        domain_age_days = (datetime.now() - creation_date).days
    except ValueError:
        domain_age_days = 0  # Unable to retrieve creation date

    # Feature 4: Presence of suspicious keywords
    suspicious_keywords = ["login", "password", "bank", "paypal", "secure"]
    suspicious_keyword_presence = any(keyword in url.lower() for keyword in
suspicious_keywords)

    return [url_length, https_presence, domain_age_days, suspicious_keyword_presence]

# Example URLs (replace with actual URLs)
real_url = "https://www.example.com"
phishing_url = "http://phishingexample.com"

# Extract features for real and phishing URLs
real_features = extract_features(real_url)
phishing_features = extract_features(phishing_url)
```

```
# Prepare dataset (example with dummy data)
X = [real_features, phishing_features]
y = [0, 1]  # 0 for real, 1 for phishing

# Split dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Evaluate model
y_pred = clf.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, y_pred))
print("Classification Report:")
print(classification_report(y_val, y_pred))

# Predict whether a new URL is phishing or not
new_url = "http://newphishingurl.com"
new_features = extract_features(new_url)
prediction = clf.predict([new_features])[0]
if prediction == 1:
    print(f"The URL {new_url} is likely a phishing website.")
else:
    print(f"The URL {new_url} is likely a legitimate website.")
```

This code demonstrates a simple approach to phishing website detection using machine learning. However, it's important to note that phishing detection is a complex problem, and this example may not capture all relevant features and nuances. Additionally, using real-world datasets and more sophisticated feature engineering techniques would likely improve the model's performance.