ASSIGNMENT 8

Question - 1

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

As a cybersecurity analyst, it is crucial to act promptly when potential security breaches are detected. Utilizing Python library Scapy will aid in examining network packets associated with the suspicious IP address. Here's how you can utilize Scapy for network packet analysis:

Explanation of Scapy for Network Packet Analysis:

Scapy is a powerful interactive packet manipulation program and library that allows for detailed network packet analysis.

It facilitates the creation, manipulation, decoding, and sending of packets through a simple and user-friendly interface. With Scapy, you can capture, dissect, and analyze network packets to gather valuable information on network activity.

Step-by-Step Process for Capturing and Analyzing Network Traffic:
Here are the steps to capture and analyze network traffic using Scapy:

- Install Scapy by running pip install scapy in your terminal.
- Use Scapy to sniff network traffic related to the suspicious IP address by creating a packet filter.

Capture network packets: Use the `sniff()` function provided by Scapy to capture network packets. You can specify filters to capture packets from a specific IP address, port, or protocol.

For example, to capture packets from the suspicious IP address, you can use:

```
packet_list = sniff(filter="host suspicious_ip_address")
```

Analyze captured packets:

Analyze the captured packets to determine the source, destination, type, and payload of each packet.

Look for any unusual patterns or anomalies in the network traffic data.

You can iterate through the captured packets to analyze their content, extract information, and look for any suspicious behavior. Scapy provides various packet dissecting functionalities to access different protocol layers and fields of the packets. For example, you can examine the source and destination IP addresses and ports, protocols, payloads, and any additional information.

Python code

```
for packet in packet_list:
    # Extract source and destination IP addresses
    source_ip = packet[IP].src
    dest_ip = packet[IP].dst
```

```
        # Extract protocol information
        protocol = packet[IP].proto

        # Extract payload
        payload = packet[IP].payload

        # Perform further analysis on extracted information
    # Look for any suspicious or anomalous behavior
        # Identify patterns, unusual protocols, large packet sizes, etc.
```

Identification of Suspicious Network Behavior: In the captured packets, look for:

- Unusual source or destination IP addresses.
- Unexpected protocols or services being used.
- Suspicious payloads that could indicate malicious intent.
- Time and frequency of network activity.

Recommendations for Mitigating Security Risks: To mitigate security risks and secure the network:

- Block the suspicious IP address and restrict network access.
- Implement intrusion detection systems and firewalls to monitor and block unauthorized network traffic.
- Regularly update network security protocols and software to prevent vulnerabilities.
- Conduct employee cybersecurity training to increase awareness and prevent social engineering attacks.

Sample Python Code for Network Packet Analysis with Scapy for network packets associated with the suspicious IP address provided with using google colab:

Python code:

```
!pip install scapy
```

```
Collecting scapy
  Downloading scapy-2.5.0.tar.gz (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 15.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
```

```
Building wheels for collected packages: scapy
  Building wheel for scapy (setup.py) ... done
  Created wheel for scapy: filename=scapy-2.5.0-py2.py3-none-any.whl
size=1444327
sha256=1cc1b1843662c4c0c6a34d6b44f684aa8a241064dddccd1fd794f
aa2e6aac3b4
  Stored in directory:
/root/.cache/pip/wheels/82/b7/03/8344d8cf6695624746311bc0d389e9d0
5535ca83c35f90241d
Successfully built scapy
Installing collected packages: scapy
    Successfully installed scapy-2.5.0

    from scapy.all import*

   def analyze_network_traffic(suspicious_ip):
        packets = sniff(filter="host " + suspicious_ip, count=10)
       # analyze and print information about sniffed packets
    for packet in packets:
       print (packet.summary())
      # Add more packet analysis logic here

suspicious_ip = "192.168.1.1"
analyze_network_traffic (suspicious_ip)

ERROR: Cannot set filter: libpcap is not available. Cannot compile filter !
ERROR:scapy.runtime:Cannot set filter: libpcap is not available. Cannot
compile filter !
Ether / IP / TCP 172.28.0.1:48664 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:48664 A
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:48664 PA / Raw
Ether / IP / TCP 172.28.0.1:48664 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:48664 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.1:48664 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:48664 A
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:48664 PA / Raw
Ether / IP / TCP 172.28.0.1:48664 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:48664 > 172.28.0.12:8080 PA / Raw
```

This code snippet demonstrates how to capture and analyze network packets related to a specific IP address using Scapy.

A python code to Network Packet Analysis with Scapy

```
!pip install scapy
```

```
Collecting scapy
  Downloading scapy-2.5.0.tar.gz (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 7.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: scapy
  Building wheel for scapy (setup.py) ... done
  Created wheel for scapy: filename=scapy-2.5.0-py2.py3-none-any.whl size=1444327 sha256=4d9478841048137b6b0376df95a4ac66a26c9680d190198a11f67a967f61c31b
  Stored in directory: /root/.cache/pip/wheels/82/b7/03/8344d8cf6695624746311bc0d389e9d05535ca83c35f90241d
Successfully built scapy
Installing collected packages: scapy
Successfully installed scapy-2.5.0
```

```
from scapy.all import*

def analyze_packets(packets):
    if iP in packet:
      src_ip=packet[IP].src
      dst_ip=packet[IP].dst
      protocol=packet[IP].proto

    print(src_ip)
    print(dst_ip)
    print(protocol)

    if TCP in packet:
      src_port=packet[TCP].sport
      dst_port=packet[TCP].dport
      flags=packet[TCP].flags
      ## Firewall Rules

    elif UDP in packet:
      src_port=packet[UDP].sport
      dst_port=packet[UDP].dport
```

```
    elif ICMP in packet:
      icmp_type=packet[ICMP].type
      icmp_code=packet[ICMP].code

    print(src_port)
    print(dst_port)
    print(flags)

sniff(prn=analyze_packets, filter="ip",count=20)
```

```
ERROR: Cannot set filter: libpcap is not available. Cannot compile filter !
ERROR:scapy.runtime:Cannot set filter: libpcap is not available. Cannot compile filter !
172.28.0.1
172.28.0.12
6
53100
8080
PA
172.28.0.12
172.28.0.1
6
8080
53100
A
172.28.0.12
172.28.0.1
6
8080
53100
PA
172.28.0.1
172.28.0.12
6
53100
8080
A
172.28.0.1
172.28.0.12
6
53100
8080
PA
```

172.28.0.1
172.28.0.12
6
53100
8080
PA
172.28.0.12
172.28.0.1
6
8080
53100
A
172.28.0.12
172.28.0.1
6
8080
53100
PA
172.28.0.1
172.28.0.12
6
53100
8080
A
172.28.0.1
172.28.0.12
6
53100
8080
PA
172.28.0.12
172.28.0.1
6
8080
53100
PA
172.28.0.1
172.28.0.12
6
53100
8080
A
172.28.0.1

172.28.0.12
6
53100
8080
PA
172.28.0.1
172.28.0.12
6
53100
8080
PA
172.28.0.12
172.28.0.1
6
8080
53100
A
172.28.0.12
172.28.0.1
6
8080
53100
PA
172.28.0.1
172.28.0.12
6
53100
8080
A
172.28.0.1
172.28.0.12
6
53100
8080
PA
172.28.0.12
172.28.0.1
6
8080
53100
PA
172.28.0.1
172.28.0.12

6
53100
8080
A
<Sniffed: TCP:20 UDP:0 ICMP:0 Other:0>

```
def analyze_network_traffic(suspicious_ip):
        packets = sniff(filter="host " + suspicious_ip, count=10)
       # analyze and print information about sniffed packets
        for packet in packets:
         print (packet.summary())
      # Add more packet analysis logic here

suspicious_ip = "192.168.1.1"
analyze_network_traffic (suspicious_ip)
```

ERROR: Cannot set filter: libpcap is not available. Cannot compile filter !
ERROR:scapy.runtime:Cannot set filter: libpcap is not available. Cannot compile filter !
Ether / IP / TCP 172.28.0.1:39844 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:39844 A
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:39844 PA / Raw
Ether / IP / TCP 172.28.0.1:39844 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:39844 > 172.28.0.12:8080 FA
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:39844 FA
Ether / IP / TCP 172.28.0.1:39844 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:50212 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:50212 PA / Raw
Ether / IP / TCP 172.28.0.1:50212 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:50212 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.1:50212 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:50212 A
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:50212 PA / Raw
Ether / IP / TCP 172.28.0.1:50212 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:50212 > 172.28.0.12:8080 PA / Raw
Ether / IP / TCP 172.28.0.1:38118 > 172.28.0.12:8080 S
Ether / IP / TCP 172.28.0.12:8080 > 172.28.0.1:38118 SA
Ether / IP / TCP 172.28.0.1:38118 > 172.28.0.12:8080 A
Ether / IP / TCP 172.28.0.1:38118 > 172.28.0.12:8080 PA / Raw

Based on the importance of investigating potential security breaches and analyzing network packets the real time implementation may require additional steps and considerations based on the specific scenario and requirements and additionally, it's important to obtain proper authorization before capturing network traffic in a production environment.

Question - 2

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

Import the necessary libraries: Start by importing the required libraries in Python, such as `requests` and `beautifulsoup4`, to handle web scraping and website analysis.

Accept the URLs: Prompt the user to enter two URLs, one for a real website and another for a potential phishing website.

Fetch website data: Use the `requests` library to fetch the HTML content of both websites.

Analyze website data: Utilize the `beautifulsoup4` library to parse the HTML content and extract relevant features for analysis. Some features you can consider are the presence of suspicious keywords, links to external domains, HTML form elements, JavaScript redirects, and presence of SSL/TLS certificates.

Apply heuristics: Define a set of heuristics or rules-based checks to identify potential phishing indicators. For example, check if the URL contains misspelled words, additional subdomains, or unusual characters.

Comparing Features between URLs:

- We'll examine the following features for each URL:
- **Domain Registration Length**: Check how long the domain has been registered.
- **SSL Certificate Validity**: Verify the SSL certificate's expiration date.
- **Presence of Specific HTML Elements**: Look for login forms (especially in non-HTTPS sites).
- **Links to External Sites**: Identify any outbound links.
- **Suspicious Email Addresses**: Detect any suspicious email addresses associated with the site.

Calculating a "Phishing Score":

Assign weights to each feature based on its significance (e.g., longer domain registration may reduce the phishing likelihood).

Calculate a score for each URL by summing up the weighted features.

Comparing Scores:

Compare the scores of both URLs.

The URL with a higher score is more likely to be a phishing sit

Build a classification model: Train a binary classification model using machine learning algorithms like logistic regression, decision tree, or random forest. Use a labelled dataset of known phishing websites and legitimate websites to create the model.

Feature engineering: Use the extracted website characteristics and indicators as input features for the classification model. You may need to reprocess and transform the data appropriately.

Train the model: Split the dataset into training and testing sets, then train the classification model using the training data.

Evaluate the model: Use the testing set to evaluate the model's performance metrics such as accuracy, precision, recall, and F1 score.

Predict phishing likelihood: Apply the trained model to the new URLs entered by the user to determine the likelihood of the second website being a phishing site.

Display the results: Provide the user with the prediction and any supporting information that led to the classification (e.g., suspicious features or rules triggered).

Phishing Website Detection with Python

```
!pip install beautifulsoup4

import requests
from bs4 import BeautifulSoup

def analyze_website(url):
  """
  Analyzes a website and extracts features to identify potential phishing attempts.

  Args:
      url: The URL of the website to analyze.

  Returns:
      A dictionary containing extracted features and a phishing likelihood score (0 - not likely, 1 - likely).
  """
  features = {}
  score = 0
```

```
  try:
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Feature 1: Check for URL age (newer domains might be suspicious)
    # (Requires additional libraries for WHOIS lookup - not implemented
here)
    # features["domain_age"] = ...

    # Feature 2: Presence of SSL certificate (HTTPS)
    features["has_https"] = "https" in url

    # Feature 3: Check for subdomain mismatch between URL and
website content (e.g., bankname.com in URL but google.com content)
    features["domain_mismatch"] = url.split("//")[1].split(".")[0] not in
soup.title.text.lower()

    # Feature 4: Check for common phishing keywords in title and content
    phishing_keywords = ["login", "password", "urgent", "verify", "account"]
    features["has_phishing_keywords"] = any(keyword in soup.text.lower()
for keyword in phishing_keywords)

    # Feature 5: Check for existence of contact information
    features["has_contact_info"] = any(tag.name in ["address", "phone",
"email"] for tag in soup.find_all())

    # Calculate score based on features
    for feature, value in features.items():
      if value and value == "likely_phishing":  # Weight features differently
based on importance
        score += 2
      elif value:
        score += 1

  except Exception as e:
    print(f"Error analyzing website: {e}")

  return features, score

def main():
  """
```

```
  Prompts user for two URLs, analyzes them, and identifies potential
phishing sites.
  """
  url1 = input("Enter URL of a real website: ")
  url2 = input("Enter URL of a suspected phishing website: ")

  features1, score1 = analyze_website(url1)
  features2, score2 = analyze_website(url2)

  print("\nReal Website Analysis:")
  print(features1)
  print(f"Phishing Likelihood Score: {score1} (Lower score indicates less
likely)")

  print("\nSuspected Phishing Website Analysis:")
  print(features2)
  print(f"Phishing Likelihood Score: {score2} (Higher score indicates more
likely)")

   if score2 > score1:
    print("\nThe suspected phishing website has a higher likelihood of
being malicious based on the analysis.")
  else:
    print("\nBased on the analysis, it's difficult to definitively classify the
suspected website.")

if __name__ == "__main__":
  main()
```

Enter URL of a real website: https://www.facebook.com
Enter URL of a suspected phishing website: https://www.facebook.co.in
Error analyzing website:
HTTPSConnectionPool(host='www.facebook.co.in', port=443): Max retries exceeded with url: / (Caused by NameResolutionError("<urllib3.connection.HTTPSConnection object at 0x79626c220580>: Failed to resolve 'www.facebook.co.in' ([Errno -2] Name or service not known)"))

Real Website Analysis:
{'has_https': True, 'domain_mismatch': True, 'has_phishing_keywords': True, 'has_contact_info': False}
Phishing Likelihood Score: 3 (Lower score indicates less likely)

Suspected Phishing Website Analysis:
{}
Phishing Likelihood Score: 0 (Higher score indicates more likely)

Based on the analysis, it's difficult to definitively classify the suspected website.

Enter URL of a real website: https://tryhackme.com
Enter URL of a suspected phishing website: https://tryyhackme.com
Error analyzing website: HTTPSConnectionPool(host='tryyhackme.com', port=443): Max retries exceeded with url: / (Caused by NameResolutionError("<urllib3.connection.HTTPSConnection object at 0x79626c290940>: Failed to resolve 'tryyhackme.com' ([Errno -2] Name or service not known)"))

Real Website Analysis:
{'has_https': True, 'domain_mismatch': False, 'has_phishing_keywords': True, 'has_contact_info': False}
Phishing Likelihood Score: 2 (Lower score indicates less likely)

Suspected Phishing Website Analysis:
{}
Phishing Likelihood Score: 0 (Higher score indicates more likely)