Assignment 8

1. Using the python library Scapy, analyse the network packets associate with the suspicious IP address provided

To analyze network packets associated with a suspicious IP address using the Scapy library in Python, you can capture packets on the network interface and filter packets based on the source or destination IP address. Below is an example code snippet to demonstrate this:

```
from scapy.all import *

# Define the suspicious IP address to filter packets

suspicious_ip = "192.168.1.1"

# Define a callback function to handle each captured packet

def packet_callback(packet):

    if IP in packet and (packet[IP].src == suspicious_ip or packet[IP].dst == suspicious_ip):

        # Print information about the suspicious packet

        print(f"Captured Packet: {packet.summary()}")

# Start capturing packets on the network interface 'eth0' and filter based on the suspicious IP address

print("Starting packet capture for suspicious IP address...")

sniff(iface='eth0', prn=packet_callback, filter=f"host {suspicious_ip}", count=10)

# 'count=10' specifies to capture and process only 10 packets, change as needed
```

Explanation of the code:

from scapy.all import *: Import all Scapy functionalities.

suspicious_ip = "192.168.1.1": Define the suspicious IP address you want to filter packets for.

packet_callback(packet): Callback function that handles each captured packet. It checks if the packet is an IP packet and if the source or destination IP matches the suspicious IP address.

sniff(iface='eth0', prn=packet_callback, filter=f"host {suspicious_ip}", count=10): Starts capturing packets on the network interface 'eth0' and filters packets based on the suspicious IP address using the BPF (Berkeley Packet Filter) syntax. The prn argument specifies the callback function to be called for each captured packet, and count=10 specifies to capture and process only 10 packets. You can change the count as needed.

Note: Replace 'eth0' with the appropriate network interface on your system. Additionally, running packet capture scripts requires administrative privileges on most systems. Also, modify the suspicious_ip variable to the actual suspicious IP address you want to analyze.

2. Step by step breakdown the process you followed to capture and analyse the network traffic

Import Scapy Library:

Import the Scapy library in Python using the following import statement:

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

Define Suspicious IP Address:

Define the suspicious IP address that you want to filter packets for. This is the IP address you suspect may be involved in suspicious network activity.

```
suspicious_ip = "192.168.1.1"  # Replace with the actual suspicious IP address
```

Define Packet Callback Function:

Define a callback function that will be called for each captured packet. This function will analyze the packets and print information about suspicious packets.

```
def packet_callback(packet):
    if IP in packet and (packet[IP].src == suspicious_ip or packet[IP].dst == suspicious_ip):
        print(f"Captured Packet: {packet.summary()}")
```

Start Packet Capture:

Start capturing packets on the specified network interface ('eth0' in this example) using the sniff function. Provide the callback function (packet_callback) as the prn argument to handle each captured packet.

Use the filter argument to specify a filter expression to capture packets only related to the suspicious IP address. The filter expression in this case is "host {suspicious_ip}".

Use the count argument to specify the number of packets to capture and process. In this example, count=10 captures and processes only 10 packets.

```
print("Starting packet capture for suspicious IP address...")
sniff(iface='eth0', prn=packet_callback, filter=f"host {suspicious_ip}", count=10)
```

Run the Script:

Run the Python script containing the above code. Ensure that you have administrative privileges to capture network packets.

The script will start capturing packets on the specified network interface ('eth0' in this example) and filter packets based on the suspicious IP address.

For each captured packet that matches the filter criteria, the callback function (packet_callback) will be called, and information about suspicious packets will be printed.

Analysis of Captured Packets:

As the script runs and captures packets, it will print information about packets where the suspicious IP address is either the source or destination IP.

Analyze the printed information to gain insights into the network traffic associated with the suspicious IP address. The printed information typically includes packet summaries, including IP addresses, protocols, port numbers, and payload details.

Modify and Customize:

Modify the script as needed to customize the packet capture and analysis process. You can change the network interface, filter criteria, callback function logic, and other parameters based on your specific requirements and use cases.

By following these steps, you can capture and analyze network traffic associated with a suspicious IP address using the Scapy library in Python.

3. Identification and interpretation of any suspicious are anomalous network behaviour observed in the captured pockets

In the context of network traffic analysis using Scapy, identification and interpretation of suspicious or anomalous network behavior typically involve examining packet information, patterns, anomalies, and potential indicators of malicious activity. Below are some common types of suspicious or anomalous network behavior that you might observe in captured packets and how to interpret them:

Unusual Port Activity:

Identify packets where the source or destination port is commonly associated with malicious services or known vulnerabilities (e.g., ports used by malware, backdoors, or exploit frameworks).

Interpretation: This could indicate attempts to exploit services, establish unauthorized connections, or perform reconnaissance on vulnerable ports.

Abnormal Protocol Usage:

Detect packets using uncommon or unexpected protocols, especially if they are not typically used in your network environment (e.g., non-standard application protocols, unusual transport layer protocols).

Interpretation: This may suggest attempts to bypass security controls, communicate covertly, or perform data exfiltration using unconventional protocols.

Large Volume of Outbound Traffic:

Notice a significant increase in outbound traffic from a specific host or to suspicious external IP addresses.

Interpretation: This could indicate command-and-control (C2) communication, data exfiltration, or compromised systems sending out large amounts of data.

Unusual Payload Patterns:

Analyze packet payloads for unusual patterns, such as encrypted data, obfuscated commands, unexpected file transfers, or known malware signatures.

Interpretation: Suspicious payload patterns may indicate malicious activities like malware infections, command execution, or data theft.

Unexpected DNS Requests:

Observe DNS requests for suspicious domain names, unusual subdomains, or frequent queries to known malicious domains.

Interpretation: This could indicate DNS tunneling, domain generation algorithms (DGAs), or communication with malicious command-and-control servers.

Unauthenticated Access Attempts:

Look for packets attempting unauthorized access to sensitive services, systems, or resources without proper authentication.

Interpretation: This behavior may indicate brute-force attacks, credential stuffing, or unauthorized access attempts by malicious actors.

Network Scans or Probes:

Identify packets that conduct scanning activities, such as port scans, service enumeration, OS fingerprinting, or vulnerability scanning.

Interpretation: This suggests reconnaissance efforts by attackers to identify potential targets, vulnerabilities, or weak points in the network.

Suspicious Traffic Patterns:

Analyze traffic patterns for spikes, bursts, unusual traffic flows, unexpected protocol combinations, or deviations from normal baseline traffic.

Interpretation: These patterns may indicate denial-of-service (DoS) attacks, network anomalies, compromised systems, or malicious activities attempting to blend in with legitimate traffic.

Unauthorized Protocol Usage:

Detect the use of protocols or services that are not approved or expected in your network environment (e.g., file sharing protocols, remote access protocols).

Interpretation: This behavior could signify policy violations, unauthorized activities, or attempts to establish unauthorized communication channels.

Repeated Failed Authentication Attempts:

Monitor packets containing repeated failed authentication attempts, especially for critical services or privileged accounts.

Interpretation: This may indicate brute-force attacks, password guessing, or credential spraying attempts to compromise user accounts.

When interpreting suspicious or anomalous network behavior observed in captured packets, it's important to consider contextual information, network policies, baseline traffic patterns, known threat intelligence, and historical data. Correlating packet analysis with other security monitoring tools, logs, and incident response processes can provide a comprehensive understanding of potential security incidents and aid in timely response and mitigation efforts.

4. Recommendations for mitigating the identified security risk and securing the network against similar threads in the future.

Mitigating identified security risks and securing the network against similar threats in the future requires a proactive approach that combines technical controls, best practices, user awareness, and continuous monitoring. Here are some recommendations for mitigating security risks and enhancing network security:

Patch Management:

Ensure all systems, applications, and network devices are regularly patched and updated with the latest security patches, bug fixes, and firmware updates. Establish a patch management process that prioritizes critical vulnerabilities and applies patches promptly.

Network Segmentation:

Implement network segmentation to divide the network into separate zones or segments based on security requirements, trust levels, and data sensitivity. Use firewalls, VLANs, access control lists (ACLs), and segmentation policies to restrict unauthorized access between segments.

Strong Authentication and Access Controls:

Enforce strong authentication mechanisms, such as multi-factor authentication (MFA), strong passwords, and secure authentication protocols (e.g., LDAP, Kerberos). Implement least privilege access controls to limit user privileges and access rights based on job roles and responsibilities.

Network Monitoring and Logging:

Deploy network monitoring tools, intrusion detection systems (IDS), intrusion prevention systems (IPS), and security information and event management (SIEM) solutions to monitor network traffic, detect anomalies, and identify potential security incidents in real time. Enable logging and log aggregation for better visibility and incident response.

Endpoint Security:

Implement endpoint security measures, such as antivirus/antimalware software, host-based firewalls, device encryption, and application whitelisting. Regularly update and patch endpoint devices, and enforce security policies for BYOD (Bring Your Own Device) and remote work environments.

Data Encryption:

Use encryption protocols (e.g., SSL/TLS, IPsec) to encrypt sensitive data in transit and at rest. Implement encryption for email communications, file transfers, databases, and sensitive information stored on servers, laptops, and mobile devices.

Employee Training and Awareness:

Conduct regular cybersecurity training sessions and awareness programs for employees to educate them about security best practices, phishing attacks, social engineering tactics, and how to recognize and report suspicious activities. Foster a culture of security awareness and accountability within the organization.

Incident Response Plan (IRP):

Develop and maintain an incident response plan (IRP) that outlines roles, responsibilities, escalation procedures, communication protocols, and incident handling workflows. Test the IRP regularly through tabletop exercises and simulated incident scenarios to ensure preparedness and effectiveness.

Vendor and Third-Party Risk Management:

Evaluate and manage risks associated with vendors, suppliers, contractors, and third-party service providers. Conduct security assessments, due diligence reviews, and contract negotiations to ensure third parties adhere to security standards and compliance requirements.

Continuous Monitoring and Auditing:

Implement continuous monitoring and auditing processes to assess network security controls, configurations, policies, and compliance with security standards (e.g., ISO 27001, NIST Cybersecurity Framework). Conduct regular security assessments, vulnerability scans, penetration testing, and audits to identify and address security gaps.

By implementing these recommendations and adopting a comprehensive approach to network security, organizations can mitigate identified security risks, enhance resilience against cyber threats, and safeguard sensitive data and assets from unauthorized access, exploitation, and disruption. Regularly review and update security measures based on evolving threats, industry trends, and lessons learned from security incidents to maintain a robust security posture.

Expected code

```
from scapy.all import *


# Define a callback function to handle each captured packet
def packet_callback(packet):
    if IP in packet:
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        proto = packet[IP].proto
        print(f"IP Packet: Source IP - {ip_src}, Destination IP - {ip_dst}, Protocol - {proto}")

        if TCP in packet:
            tcp_sport = packet[TCP].sport
            tcp_dport = packet[TCP].dport
            print(f"TCP Segment: Source Port - {tcp_sport}, Destination Port - {tcp_dport}")

        elif UDP in packet:
            udp_sport = packet[UDP].sport
            udp_dport = packet[UDP].dport
            print(f"UDP Datagram: Source Port - {udp_sport}, Destination Port - {udp_dport}")

        elif ICMP in packet:
            icmp_type = packet[ICMP].type
            icmp_code = packet[ICMP].code
            print(f"ICMP Message: Type - {icmp_type}, Code - {icmp_code}")
```

```
# Start capturing packets on the network interface 'eth0'
print("Starting packet capture...")
sniff(iface='eth0', prn=packet_callback, count=10)
# 'count=10' specifies to capture and process only 10 packets, change as needed
```

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

Ans: To develop a phishing website detection system in Python, we can use various website characteristics such as URL structure, domain age, SSL certificate presence, content analysis, etc. Below is an example code outlining the procedure to accept two web URLs, analyze their data, and identify the phishing site based on certain criteria.

Expected code

```
import requests
from bs4 import BeautifulSoup
import re
import whois
import datetime

def get_html_content(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            return response.text
        else:
            print(f"Error: Unable to fetch HTML content from {url}")
            return None
    except Exception as e:
        print(f"Error: {e}")
        return None

def extract_domain_age(url):
    try:
        domain = re.search(r"https?://(www\.)?([^/]+)/?", url).group(2)
        domain_info = whois.whois(domain)
        creation_date = domain_info.creation_date
        if type(creation_date) is list:
```

```
            creation_date = creation_date[0]
        if creation_date:
            return (datetime.datetime.now() - creation_date).days
        else:
            return None
    except Exception as e:
        print(f"Error extracting domain age: {e}")
        return None

def analyze_website(url):
    features = {}
    html_content = get_html_content(url)
    if html_content:
        soup = BeautifulSoup(html_content, 'html.parser')

        # Check for HTTPS
        features['https'] = 1 if url.startswith("https") else 0

        # Check for presence of form tags (common in phishing sites)
        features['form_tags'] = len(soup.find_all('form'))

        # Check for domain age
        features['domain_age_days'] = extract_domain_age(url)

        # Check for presence of suspicious keywords in URL or content
```

```
        keywords = ['login', 'password', 'account', 'bank', 'verify', 'secure']
        for keyword in keywords:
            features[f'keyword_{keyword}'] = 1 if keyword in url.lower() else 0
            features[f'content_{keyword}'] = 1 if soup.find(text=re.compile(keyword, re.IGNORECASE)) else 0

        return features
    else:
        return None

def identify_phishing_site(real_url, phishing_url):
    real_features = analyze_website(real_url)
    phishing_features = analyze_website(phishing_url)

    if not real_features or not phishing_features:
        print("Error: Unable to analyze website features.")
        return None

    # Example criteria for phishing detection (you can modify or add more criteria)
    phishing_score = 0
    if phishing_features['https'] == 0:
        phishing_score += 1
    if phishing_features['form_tags'] > 0:
        phishing_score += 1
    if phishing_features['domain_age_days'] and phishing_features['domain_age_days'] < 30:
        phishing_score += 1
```

```
    if phishing_features['keyword_password'] == 1 or phishing_features['content_password'] == 1:
        phishing_score += 1

    if phishing_score >= 2:  # Adjust the threshold as needed
        return phishing_url
    else:
        return real_url

# Example usage
real_website_url = "https://www.google.com"
phishing_website_url = "https://example.com/phishing"
phishing_detected = identify_phishing_site(real_website_url, phishing_website_url)
if phishing_detected:
    print(f"The detected phishing website is: {phishing_detected}")
else:
    print("No phishing website detected.")
```