actividad-11

May 1, 2024

1 Diseño de red segura

2 1.Configuración de VPN con Python

Una empresa necesita diseñar una red segura que conecte tres sucursales ubicadas en diferentes ciudades utilizando tecnología WAN y LAN. La empresa maneja datos confidenciales y requiere que la comunicación entre sucursales sea cifrada.

[ ]:
```
!pip install paramiko
```

[ ]:
```
import paramiko


def connect_to_router(hostname, username, password, timeout=10):

    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    try:
        client.connect(hostname, username=username, password=password,␣
 ↪timeout=timeout)
        return client
    except (TimeoutError, paramiko.SSHException) as e:
        print(f"Error de coneccion: {e}")
        raise

def configure_ipsec_vpn(client, peer_ip, local_network, remote_network):

    commands = [
        'crypto isakmp policy 10',
        'encr aes 256',
        'authentication pre-share',
        'group 5',
        f'crypto isakmp key mysharedsecret address {peer_ip}',
        'crypto ipsec transform-set myset esp-aes 256 esp-sha-hmac',
        'crypto map mymap 10 ipsec-isakmp',
        f'set peer {peer_ip}',
        'set transform-set myset',
```

```
        'match address 100',
        f'access-list 100 permit ip {local_network} {remote_network}',
        'interface g0/0',
        'crypto map mymap',
        'end'
    ]

    for command in commands:
        stdin, stdout, stderr = client.exec_command(command)
        print(stdout.read().decode())

    client.close()

hostname = '192.168.1.1'
username = 'admin'
password = 'password'

try:
    client = connect_to_router(hostname, username, password)
    configure_ipsec_vpn(client, '192.168.2.1', '192.168.1.0 255.255.255.0',␣
 ↪'192.168.3.0 255.255.255.0')
except (TimeoutError, paramiko.SSHException) as e:
    print(f"Se produjo un error: {e}")
```

3 2.Optimización de protocolos y caché

Una empresa de streaming de video experimenta interrupciones frecuentes en la entrega de contenido a los clientes. La infraestructura actual utiliza multicast para distribuir contenido, pero aún enfrenta problemas de latencia y pérdida de paquetes.

[ ]:
```
class VideoCache:
    def __init__(self):
        self.cache = {}

    def get_video(self, video_id):
        if video_id in self.cache:
            print(f"Video {video_id} recuperado de la caché")
            return self.cache[video_id]
        else:
            print(f"Video {video_id} no está en la caché, descargando...")
            video_data = self.download_video(video_id)
            self.cache[video_id] = video_data
            return video_data

    def download_video(self, video_id):
        return f"Datos del video para {video_id}"
```

```
cache = VideoCache()
video = cache.get_video("video1234")
print(video)
video = cache.get_video("video1234")
print(video)
```

Implementación de anycast con Python

Simularemos el uso de anycast para dirigir las solicitudes de los usuarios al servidor de caché más cercano utilizando Python. Este ejemplo es conceptual, ya que la implementación real de anycast se manejaría a un nivel más bajo en la red

[ ]:
```
import random

class AnycastService:
    def __init__(self):
        self.servers = ['192.168.1.1', '192.168.2.1', '192.168.3.1']

    def get_nearest_server(self, user_ip):
        return random.choice(self.servers)


anycast = AnycastService()
nearest_server = anycast.get_nearest_server("192.168.1.100")
print(f"servidor cercano {nearest_server}")
```

4 3.Simulación de ataque y respuesta

Eres el administrador de seguridad de una red corporativa y has notado un aumento en el tráfico ARP anómalo. Sospechas que esto podría ser parte de un ataque de ARP spoofing, donde un atacante podría estar intentando interceptar la comunicación entre dos partes para robar o modificar datos transmitidos

[ ]:
```
!pip install scapy
```

[ ]:
```
from scapy.all import ARP, sniff, getmacbyip

def detect_arp_spoofing(packet):
    if packet.haslayer(ARP):
        if packet[ARP].op == 2:  # Es un ARP response (ARP reply)
            try:
                real_mac = getmacbyip(packet[ARP].psrc)
                response_mac = packet[ARP].hwsrc
                if real_mac != response_mac:
```

```
                    print(f"[ALERTA] Se ha detectado ARP Spoofing: {packet[ARP].
↪psrc} ha sido suplantado!")
            except IndexError:
                pass

def sniff_network(interface):
    sniff(iface=interface, store=False, prn=detect_arp_spoofing, filter="arp")


sniff_network('eth0')
```

Automatización y monitorización

[ ]:
```
!pip install netmiko
```

[ ]:
```
from netmiko import ConnectHandler, NetmikoTimeoutException

def enable_dai(switch_details):
    commands = [
        'ip arp inspection vlan 1-100',
        'interface range fa0/1-24',
        'ip arp inspection limit rate 100'
    ]

    try:
        with ConnectHandler(**switch_details) as switch:
            output = switch.send_config_set(commands)
        print(output)
    except NetmikoTimeoutException as e:
        print(f"Conección fallida: {e}")

switch_details = {
    'device_type': 'cisco_ios',
    'host': '192.168.1.1',
    'username': 'admin',
    'password': 'password',
}

enable_dai(switch_details)
```

5 4.Análisis y diseño de red Peer-to-Peer (P2P)

Un startup tecnológico desea implementar una red P2P robusta para permitir el intercambio eficiente de recursos computacionales entre usuarios distribuidos geográficamente. Esta red debe ser capaz de manejar intercambios dinámicos de archivos, distribución de carga, y debe incorporar medidas de seguridad para prevenir accesos no autorizados.

Implementación de la Red P2P con Python

[ ]:
```python
import os
import base64
from cryptography.fernet import Fernet

def generate_key():
    return Fernet.generate_key()

def verify_key(key, password):
    try:
        fernet = Fernet(key)
        fernet.decrypt(password.encode())
        return True
    except Exception as e:
        print(f"Error de vereficacion de clave: {e}")
        return False

def encrypt_message(message, key):
    fernet = Fernet(key)
    encrypted_message = fernet.encrypt(message.encode())
    return encrypted_message

def decrypt_message(encrypted_message, key):
    fernet = Fernet(key)
    decrypted_message = fernet.decrypt(encrypted_message).decode()
    return decrypted_message


key = generate_key()
print("clave generada.")


password = "secure_password"
print("Verificación:", verify_key(key, password))


message = "Mensaje secreto"
encrypted = encrypt_message(message, key)
print("mensaje cifrado:", encrypted)
decrypted = decrypt_message(encrypted, key)
print("mensaje:", decrypted)
```

Seguridad en la Red P2P

Implementar autenticación y cifrado en la comunicación entre nodos para asegurar que solo los usuarios autorizados puedan unirse y que la transferencia de archivos sea segura.

[ ]:
```python
import socket
import threading

class Peer:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.peers = []
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.bind((self.host, self.port))
        self.server.listen(5)
        print(f"Nodo iniciado en {self.host}:{self.port}")
        threading.Thread(target=self.accept_connections).start()

    def accept_connections(self):
        while True:
            client, address = self.server.accept()
            print(f"Conectado con {address[0]}:{address[1]}")
            self.peers.append(client)
            threading.Thread(target=self.handle_client, args=(client,)).start()

    def handle_client(self, client):
        while True:
            try:
                data = client.recv(1024)
                if data:
                    print(f"Recibido: {data.decode()}")
                    self.broadcast_data(data, client)
            except Exception as e:
                print(f"Error: {e}")
                client.close()
                break

    def broadcast_data(self, data, sender):
        for peer in self.peers:
            if peer is not sender:
                peer.send(data)

    def connect_to_peer(self, host, port):
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((host, port))
        self.peers.append(client)
        print(f"Conectado al par en {host}:{port}")

node2 = Peer('127.0.0.1', 600)
node.connect_to_peer('127.0.0.1', 600)
```