Problema 1: Diseño de red segura

Escenario

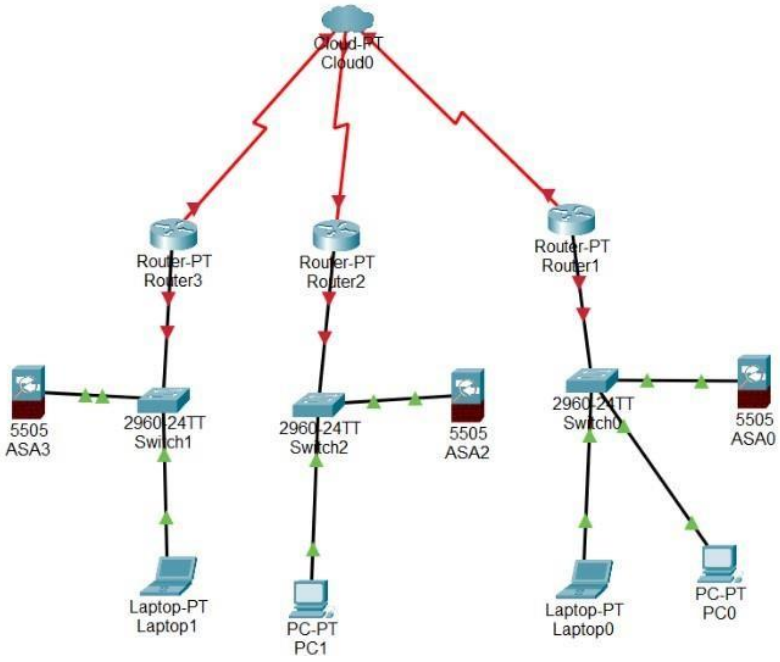
Una empresa necesita diseñar una red segura que conecte tres sucursales ubicadas en diferentes ciudades utilizando tecnología WAN y LAN. La empresa maneja datos confidenciales y requiere que la comunicación entre sucursales sea cifrada.

Preguntas

1. ¿Qué tipo de tecnología de WAN utilizarías para conectar las sucursales y por qué? Para conectar las sucursales ubicadas en diferentes ciudades, optaríamos por MPLS (Multiprotocol Label Switching). MPLS proporciona un alto nivel de seguridad y calidad de servicio (QoS), lo que es crucial para una empresa que maneja datos confidenciales y requiere comunicación cifrada entre sucursales. Además, MPLS permite la creación de redes privadas virtuales (VPN) fácilmente configurables, lo que garantiza la privacidad y seguridad de los datos transmitidos.

2. Describe cómo implementarías el cifrado en la red. ¿Qué tipos de claves y protocolos utilizarías?
   Para asegurar una comunicación cifrada de extremo a extremo, utilizaríamos el protocolo IPsec (Internet Protocol Security). IPsec proporciona autenticación, integridad de datos y confidencialidad mediante la encapsulación de paquetes IP en un túnel seguro. Utilizaría claves precompartidas para la autenticación y el cifrado, junto con el algoritmo AES (Advanced Encryption Standard) para garantizar una alta seguridad.

3. Dibuja una topología de red que incluya dispositivos como routers, switches, y firewalls. Explica la función de cada dispositivo en tu diseño. Puedes utilizar PacketTracer



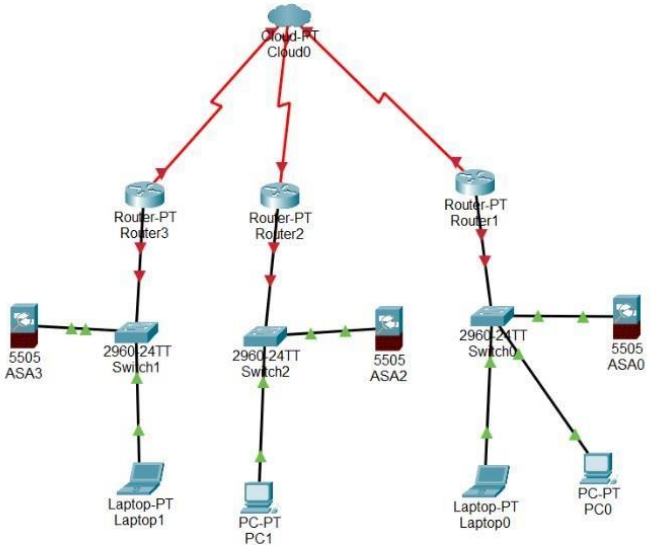
4. ¿Cómo garantizarías la integridad y autenticidad de los datos transmitidos entre las sucursales? Detalla el uso de checksums o CRC.

Para garantizar la integridad y autenticidad de los datos transmitidos entre las sucursales, implementaría checksums o CRC (Cyclic Redundancy Check) en cada router. Estos mecanismos de verificación de integridad permiten detectar cualquier alteración en los datos durante la transmisión y aseguran que los datos recibidos sean los mismos que los enviados.

Parte 1: Diseño de topología de red

Preguntas

1. Dibuja una topología de red para este escenario que incluya los dispositivos de red necesarios en cada sucursal.



2. Explica cómo cada dispositivo contribuye a la seguridad y eficiencia de la red.

    - **Router**: Actúa como el punto de conexión a la red WAN/MPLS, gestionando el tráfico entrante y saliente hacia y desde otras sucursales. Su función principal es enrutar los paquetes de datos entre las redes LAN de las sucursales y la red WAN, proporcionando conectividad segura y eficiente.

    - **Switch**: Conecta los dispositivos de la red local de cada sucursal, permitiendo la comunicación entre ellos a través de la conmutación de paquetes. Los switches segmentan el tráfico de la red, mejorando la eficiencia al limitar la difusión de datos únicamente a los dispositivos destinatarios. Además, algunos switches pueden implementar funciones de seguridad como VLANs para aislar el tráfico y prevenir accesos no autorizados.

    - **Firewall**: Protege la red local de cada sucursal contra accesos no autorizados desde Internet y aplica políticas de seguridad. Actúa como una barrera entre la red interna y externa, filtrando el tráfico según reglas predefinidas para permitir o denegar el acceso a determinados servicios o recursos. Los firewalls son fundamentales para garantizar la seguridad de la red al bloquear amenazas externas y prevenir ataques maliciosos.

Parte 2: Configuración de VPN con Python

Utilizando la biblioteca paramiko de Python, escribe un script que configure una VPN en los routers de cada sucursal. Supondremos que los routers son dispositivos Cisco y que el script debe configurar automáticamente la VPN utilizando IPsec.

```
# Código Python para configurar la VPN en los routers de cada sucursal import
paramiko

# Función para conectar al router def
connect_to_router(hostname, username, password):
client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect(hostname, username=username, password=password)
return client

# Función para configurar la VPN def configure_ipsec_vpn(client,
peer_ip, local_network, remote_network):     commands = [
        'crypto isakmp policy 10',
        'encr aes 256',
        'authentication pre-share',
        'group 5',
        'crypto isakmp key mysharedsecret address ' + peer_ip,
        'crypto ipsec transform-set myset esp-aes 256 esp-sha-hmac',
        'crypto map mymap 10 ipsec-isakmp',
        'set peer ' + peer_ip,
        'set transform-set myset',
        'match address 100',
        '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()

# Ejemplo de uso hostname
= '192.168.1.1' username
= 'admin' password =
'password'
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')
```

Preguntas adicionales

3. **¿Cómo implementarías el cifrado de extremo a extremo además de la VPN? Considera el uso de claves públicas y privadas.**
   Además de la VPN, implementaría el cifrado de extremo a extremo utilizando claves públicas y privadas. Cada sucursal tendría un par de claves única. Cuando un dispositivo de una sucursal envía datos a otra sucursal, cifraría los datos con la clave pública de la sucursal receptora. Así, solo la sucursal receptora, con su clave privada correspondiente, podría descifrar los datos. Esto proporciona una capa adicional de seguridad sobre la VPN.

4. **Proporciona un esquema para implementar un sistema robusto de logs y monitoreo de la red utilizando herramientas modernas de gestión de red. ¿Cómo podría Python automatizar la recopilación y análisis de logs?**
   Para implementar un sistema de logs y monitoreo de la red, utilizaría herramientas como ELK Stack (Elasticsearch, Logstash, Kibana) o Splunk. Estas herramientas permiten recopilar, almacenar, visualizar y analizar logs de manera eficiente. Python podría automatizar la recopilación y análisis de logs mediante el uso de bibliotecas como `elasticsearch`, `logstash`, `kibana`, `splunk-sdk`, entre otras, para interactuar con estas herramientas y realizar tareas específicas de monitoreo y análisis de logs de forma programática.

Integrantes:

- Huanca Hampuero Lila Zaray
- Gavino Isidro Michael Richard
- Manosalva Peralta Yojan Alexander

Problema 2: Optimización de protocolos y caché

Escenario: 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.

Preguntas:

1. Explica cómo mejorarías el rendimiento utilizando técnicas de caché de red. ¿Dónde colocarías estos cachés y por qué?
   Para la mejora del rendimiento implementaríamos los cachés cerca de los puntos de acceso donde los clientes tienen una mayor demanda con el uso de internet. Esto disminuiría la latencia y proporcionar un acceso rápido a los datos.

2. ¿Cómo afecta el protocolo de transporte al rendimiento del streaming de video? Considera TCP vs UDP y justifica tu elección.
   En el transporte UDP tiene mayor control de flujo y menor latencia que el TCP, además UDP es mas adecuado para los usos de mayor transmisión de datos en tiempo real como en nel streaming, es mejor la velocidad que la integridad de los datos, por ello no garantiza la entrega de paquetes en orden, pero se puede implementar un método de retransmisión para mitigar esto.

3. Propone una solución usando Anycast para optimizar la entrega de contenido. ¿Cómo funcionaría en este contexto?
   En el contenido de streaming, Anycast se puede utilizar para dirigir las solicitudes de los usuarios al servidor cache más cercano, esto ayuda con la velocidad de entrega de los datos y disminuye la latencia de los servidores principales, mejorando la experiencia del usuario.

4. Desarrolla un modelo simplificado para calcular el efecto de la caché en la reducción de latencia.

Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Mejora del rendimiento utilizando técnicas de caché de red.
2. Elección y configuración del protocolo de transporte adecuado para reducir la latencia y mejorar la confiabilidad.
3. Implementación de Anycast para optimizar la entrega de contenido a los usuarios.
4. Simulación y automatización utilizando Python.

Parte 1: Implementación de caché de red con Python

Usaremos Python para simular un sencillo sistema de caché que pueda almacenar y recuperar videos solicitados frecuentemente para reducir la latencia y la carga en el servidor principal.

Código Python:

```
class VideoCache: def
   __init__(self):

      self.cache = {}

   def get_video(self, video_id):

      if video_id in self.cache: print(f"Video {video_id}
         retrieved from cache") return
         self.cache[video_id]

      else:

         print(f"Video {video_id} not in cache, downloading...") video_data =
         self.download_video(video_id) self.cache[video_id] = video_data
         return video_data

   def download_video(self, video_id):
```

```
        # Simula la descarga del video desde un servidor remoto return
        f"Video data for {video_id}"

# Ejemplo de uso cache = VideoCache()
video = cache.get_video("video1234")

print(video) # Primera vez descarga, segunda vez desde caché video =
cache.get_video("video1234")
```

Parte 2: Selección del protocolo de transporte

Exploraremos cómo usar UDP en lugar de TCP para mejorar la eficiencia de la transmisión de video, debido a la menor sobrecarga de UDP.

Discusión:

- Explica las ventajas de usar UDP sobre TCP para streaming de video, considerando las características de ambos protocolos.
  UDP tiene menor latencia en cuanto al TCP, ya que esto tiene menor control de flujo, transmisión de paquetes y el establecimiento de conexión.

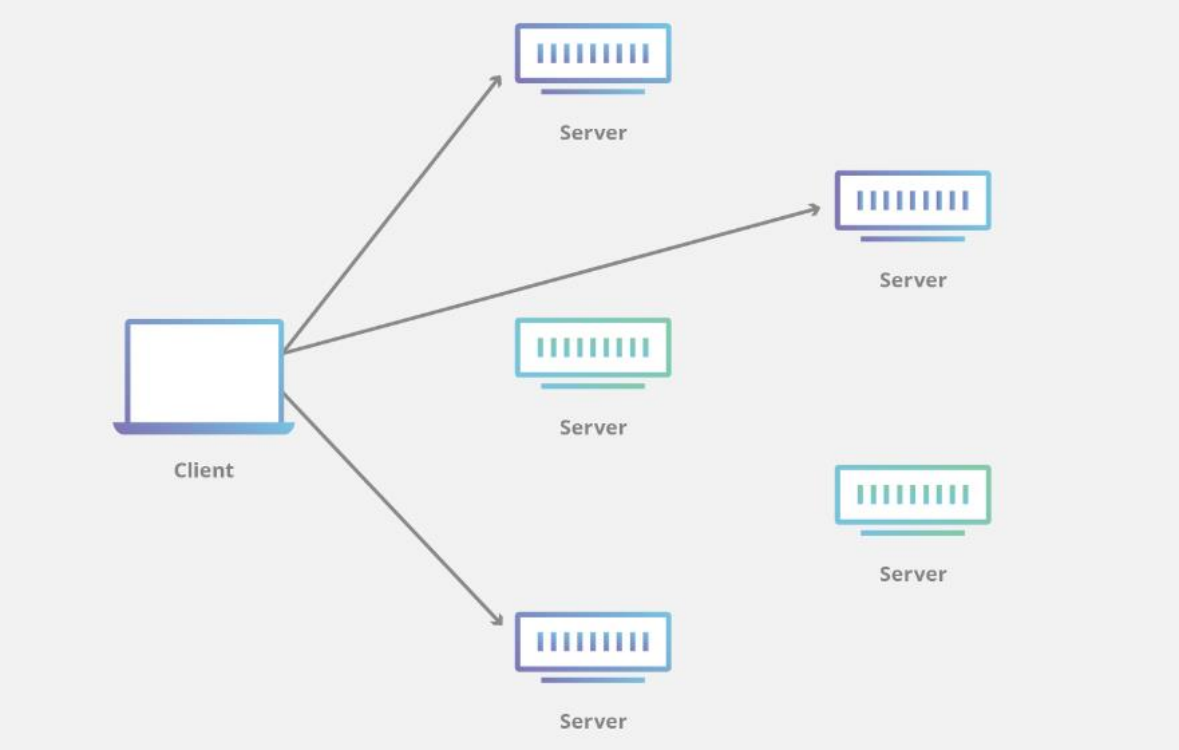
  UDP tiene mayor transmisión rápida de datos en tiempo real, priorizando la velocidad.

- Analiza los posibles problemas de confiabilidad y orden de llegada de los paquetes y cómo mitigarlos.

  UDP no garantiza que los paquetes lleguen tampoco como en el orden en el que fue enviado. Para solucionar se implementaría mecanismos de retransmisión dentro de la aplicación y agregar números de secuencia para la corrección de errores en el flujo y reordenarlos en el lado del receptor.

Parte 3: 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):

        # Simula la selección del servidor más cercano (simplificado) return
        random.choice(self.servers)

# Ejemplo de uso anycast =
AnycastService()

nearest_server = anycast.get_nearest_server("192.168.1.100") print(f"Nearest
server for user is {nearest_server}")
```

Parte 4: Monitorización y análisis

Proponer un sistema de monitoreo y análisis para evaluar el rendimiento de la red y la eficacia de las estrategias implementadas, usando herramientas como Wireshark para analizar el tráfico de red.

- Usa wireshark para capturar paquetes de red y analiza el tráfico específico de video para identificar patrones de uso y posibles cuellos de botella.

- Análisis de tráfico: Examinar el tráfico para identificar los patrones que generan los cuellos de botella.

- Cuellos de botella: Localizar las áreas de mayor tráfico de datos y mayor latencia en la red.

- Optimización: Configurar los cachés de la red, los protocolos de transporte y enrutamiento, además, de la implementación de Anycast para mejorar el rendimiento y la confiabilidad en este contexto del streaming de video.

Problema 3: Simulación de ataque y respuesta

Escenario: 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.

Preguntas:

1. Describe cómo identificarías si estas transmisiones ARP son realmente maliciosas.
   Podemos hacer una comparación de las direcciones MAC reales con las direcciones que se envían de respuestas ARP.

```
from scapy.all import ARP, sniff
from scapy.layers.l2 import getmacbyip


def detect_arp_spoofing(packet):
    if packet.haslayer(ARP):
        if packet[ARP].op == 2:  # is it an ARP response (ARP reply)?
            try:
                real_mac = getmacbyip(packet[ARP].psrc)
                response_mac = packet[ARP].hwsrc
                if real_mac != response_mac:
                    print(f"[ALERT] ARP Spoofing Detected: {packet[ARP].psrc} has been spoofed!")
            except IndexError:
                pass
```

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

# Ejemplo
sniff_network('eth0')
```

2. ¿Qué medidas tomarías para mitigar el ataque una vez confirmado?
   Una vez confirmado el ataque podemos implementar la seguridad el swicht con firewall y seguridad de los puertos. Para habilitar la función de seguridad como Dynamic ARP Inspección (DAI) en los switchs y la configuración de los puertos de switchs y el numero de acceso a los puertos MACs respectivamente.

3. Explica cómo un switch y un firewall pueden configurarse para prevenir futuros ataques de este tipo.
   Los datos del switch son enviados al firewall para que sean filtradas y verificar que no tengan errores así como verificar las infiltraciones que puedan haber.

4. Formula un plan para educar a los empleados sobre medidas de seguridad que pueden tomar para reducir el riesgo de ataques futuros.

Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Detección de ARP Spoofing: Implementar un sistema de detección para identificar posibles ataques.
2. Mitigación del Ataque: Desarrollar un método para mitigar o detener el ataque una vez detectado.
3. Automatización y Monitorización: Usar Python para automatizar la detección y respuesta, y monitorear continuamente la red para futuros ataques.
4. Educación de Empleados: Proporcionar un plan para educar a los empleados sobre cómo pueden ayudar a reducir el riesgo de futuros ataques.

Parte 1: Detección de ARP Spoofing con Python

Utilizaremos Python y la biblioteca **Scapy,** que es muy potente para la manipulación y análisis de paquetes de red, para detectar anomalías en los mensajes ARP.

Código Python para la Detección de ARP Spoofing:

```
from scapy.all import ARP, sniff def

detect_arp_spoofing(packet):

    if packet.haslayer(ARP):

        if packet[ARP].op == 2: # is it an ARP response (ARP reply)?

            try:

                real_mac = getmacbyip(packet[ARP].psrc)

                response_mac = packet[ARP].hwsrc if

                real_mac != response_mac:

                    print(f"[ALERT] ARP Spoofing Detected: {packet[ARP].psrc} has been
spoofed!") except

            IndexError:

                # Unable to find the MAC address for the IP address in the ARP response pass


def sniff_network(interface): sniff(iface=interface, store=False,

    prn=detect_arp_spoofing, filter="arp")


# Ejemplo de uso

sniff_network('eth0')
```

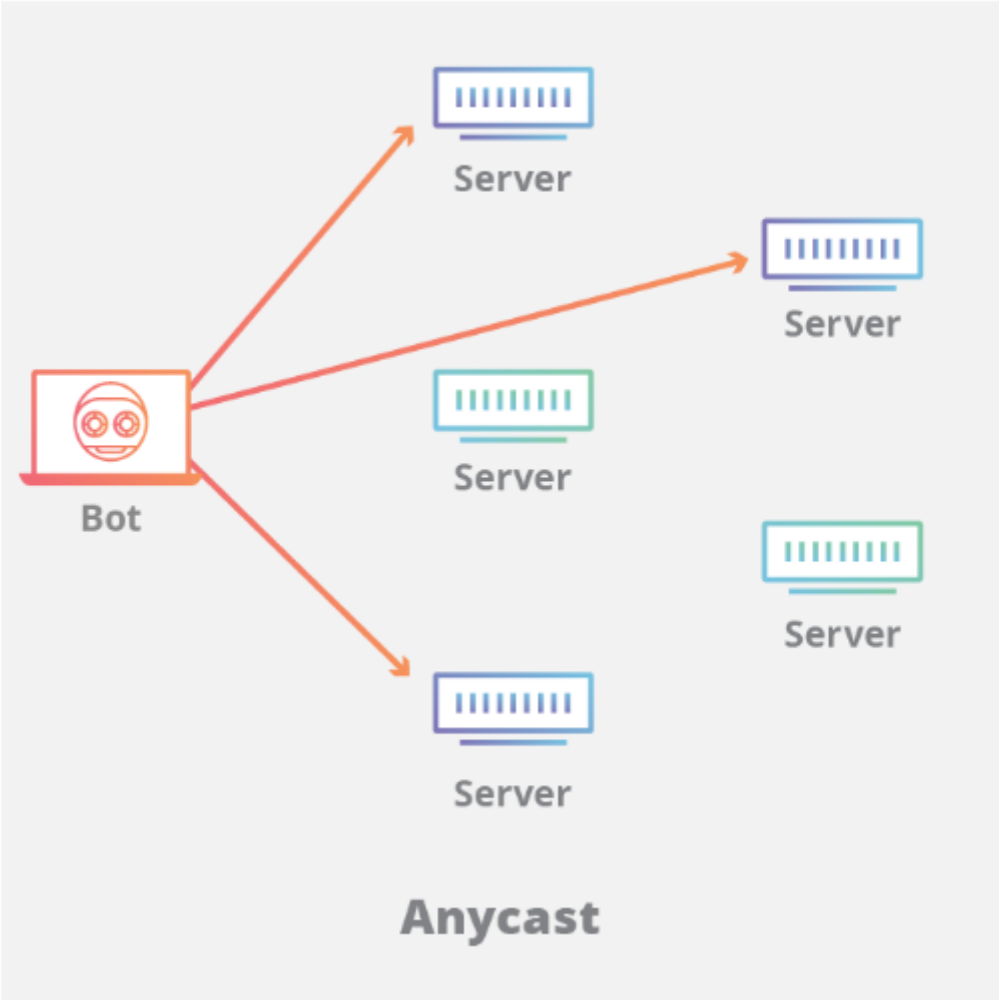
Este script escucha en la interfaz de red especificada para paquetes ARP y compara la dirección MAC real con la dirección MAC que envía respuestas ARP. Si no coinciden, se detecta un ataque de spoofing.

Parte 2: Mitigación del ataque

Una vez detectado el ataque, se puede implementar una estrategia para mitigar el daño y prevenir futuras ocurrencias, como la reconfiguración de switches para habilitar ciertas características de seguridad.

Discusión de Estrategias de Mitigación:

- Seguridad en el Switch: Habilitar funciones de seguridad como Dynamic ARP Inspection (DAI) en los switches.
- Uso de Seguridad de Puerto: Configurar la seguridad de puerto en los switches para limitar el número de MACs por puerto y prevenir posibles ataques.



Parte 3: Automatización y monitorización

Podemos usar Python para crear scripts que configuran automáticamente los switches para habilitar estas funciones de seguridad, usando bibliotecas como Netmiko para gestionar dispositivos de red.

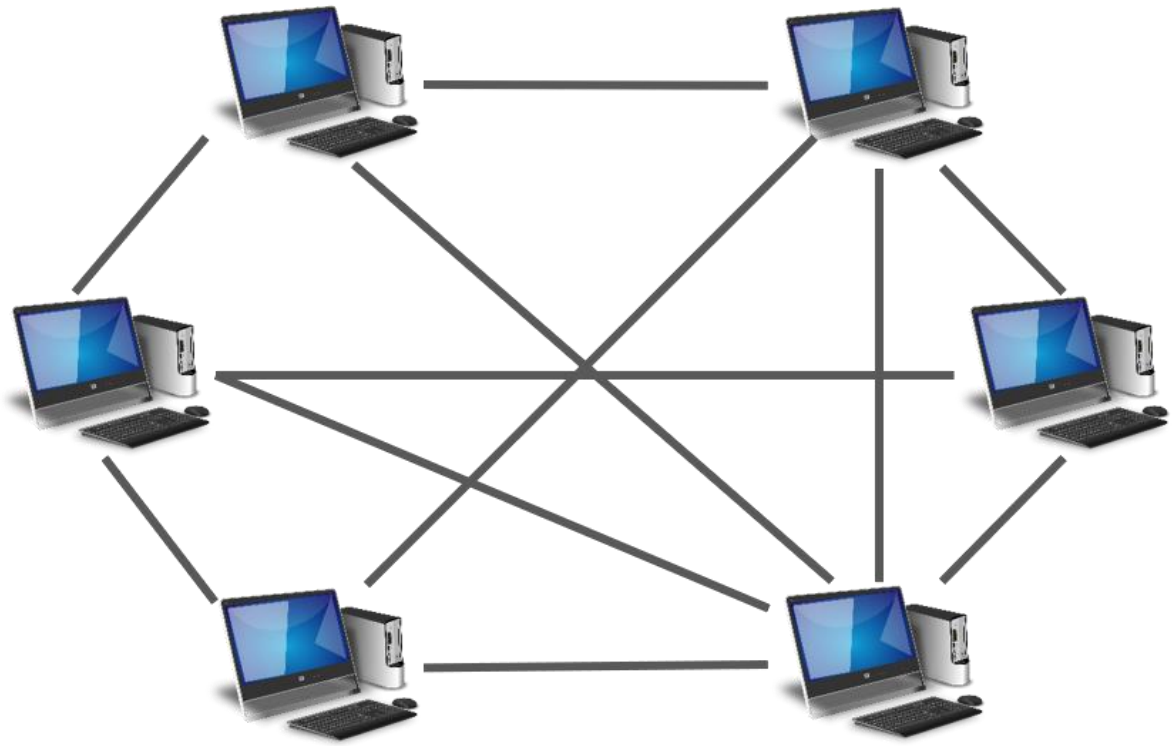
```
from netmiko import ConnectHandler

def enable_dai(switch_details):

    commands = [

        'ip arp inspection vlan 1-100',

        'interface range fa0/1-24',

        'ip arp inspection limit rate 100'

    ]

    with ConnectHandler(**switch_details) as switch: output

        = switch.send_config_set(commands)


        print(output)



# Ejemplos de uso

switch_details = {

    'device_type': 'cisco_ios',

    'host': '192.168.1.1',

    'username': 'admin',

    'password': 'password',

}

enable_dai(switch_details) Opcional:
```

Enfocarse en la importancia de la seguridad en la red y cómo las acciones de los empleados pueden afectarla.

- Indicar a tus compañeros a reconocer correos electrónicos sospechosos y otras posibles fuentes de ataques.

- No guardar contraseñas en servicios públicos y no abrir correos sospechosos.
- Informarse bien de las políticas se seguridad y privacidad.
- Mantener actualizado los sistemas y aplicar parches de seguridad para prevenir los ataques.

Problema 4: Análisis y diseño de red Peer-to-Peer (P2P)



Escenario: Una startup tecnológica 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.

Preguntas:

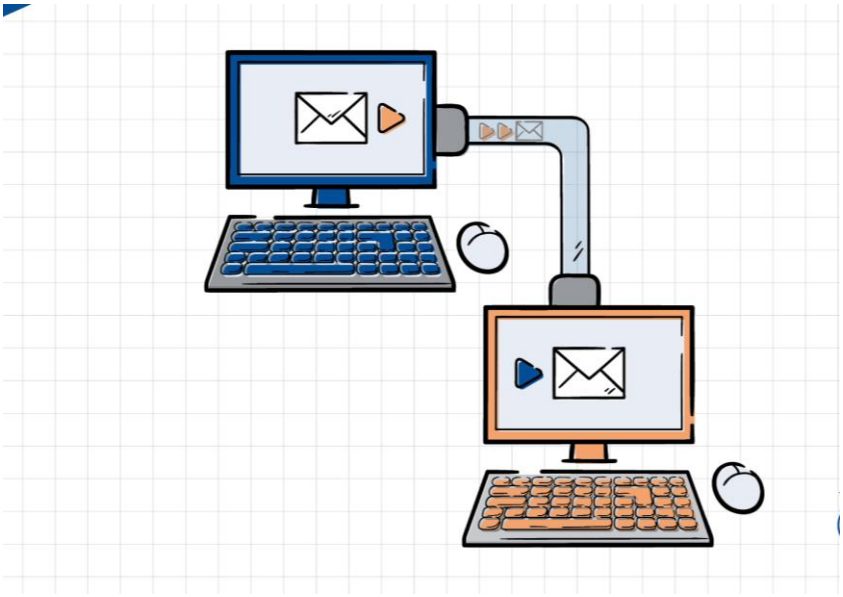
1. Describe cómo se diferenciaría esta red P2P de una red cliente-servidor en términos de diseño de red y topología.
2. ¿Qué protocolos específicos usarías para gestionar las comunicaciones y el intercambio de archivos en esta red?
3. Analiza los posibles problemas de seguridad asociados con una red P2P y propone soluciones para mitigar estos riesgos.
4. Evalúa el impacto de incorporar nodos que actúan tanto como clientes como servidores. ¿Cómo gestionarías el balanceo de carga?

Para tu presentación y código a presentar puedes utilizar:

Requisitos:

1. Implementación de la Red P2P: Desarrollar un modelo de red P2P utilizando Python que permita a los nodos unirse, compartir recursos y desconectarse de manera dinámica.
2. Gestión de Recursos y Distribución de Carga: Asegurar que los recursos se compartan de manera eficiente y que la carga se distribuya equitativamente entre los nodos.
3. Seguridad: Implementar mecanismos de seguridad para autenticar a los usuarios y cifrar los intercambios de archivos.
4. Automatización y Simulación: Utilizar Python para simular el comportamiento de la red y automatizar tareas comunes de gestión de la red.

Parte 1: Implementación de la Red P2P con Python



Utilizaremos Python para simular una red P2P básica donde los nodos pueden unirse, compartir archivos y salir de la red. Para esto, usaremos sockets para la comunicación entre nodos.

Código Python para la simulación de Red P2P:

```
import socket

import threading



class Peer:

    def __init__(self, host, port):
```

```
    self.host = host self.port = port self.peers = [] # List to keep track of peers

    self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    self.server.bind((self.host, self.port)) self.server.listen(5)

    print(f"Node started on {self.host}:{self.port}")

    threading.Thread(target=self.accept_connections).start()


def accept_connections(self):

    while True:

        client, address = self.server.accept() print(f"Connected with

        {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"Received: {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"Connected
        to peer at {host}:{port}")

# ejemplos de uso
node = Peer('127.0.0.1', 5000)
node.connect_to_peer('127.0.0.1', 6000)
```

Parte 2: Gestión de recursos y distribución de carga

Discusión sobre cómo optimizar la distribución de recursos en la red, utilizando algoritmos de distribución de carga y equilibrio de carga basados en la disponibilidad de recursos de cada nodo.

Estrategias de equilibrio de carga:

- Implementar un algoritmo que asigne más carga a los nodos con más recursos disponibles.
- Usar un algoritmo round-robin para asegurar que todos los nodos compartan de manera equitativa la carga de trabajo.

Parte 3: 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.

import hashlib import os def generate_key(password): salt = os.urandom(16) key =

hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 100000) return salt +

key def verify_key(stored_key, provided_password):

salt = stored_key[:16]

key = stored_key[16:]

new_key =hashlib.pbkdf2_hmac('sha256', provided_password.encode('utf-8'), salt,

100000) return

new_key == key

password = "secure_password" key =

generate_key(password) print("Key generated

successfully.") print("Verification:",

verify_key(key, password))

Monitorización de una red

P2P import time def

monitor_peers(peers): while

True: for peer in peers:

try:

peer.send(b'ping')

response = peer.recv(1024) if response != b'pong': raise

Exception("Peer is not responding correctly.") except

Exception as e:

print(f"Peer {peer.getpeername()} failed: {e}")

peers.remove(peer) time.sleep(60) # Check every minute

In the Peer class, handle 'ping' messages

def handle_client(self, client):

while True:

try:

data = client.recv(1024)

if data:

if data == b'ping':

client.send(b'pong')

else:

```
                print(f"Received: {data.decode()}")
        self.broadcast_data(data, client) except
        Exception as e:
            print(f"Error:
            {e}")
            client.close()
            break
```

Seguridad en la red:

```
import hashlib
import os


def generate_key(password):
    salt = os.urandom(16)
    key = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 100000)
```

```
    return salt + key



def verify_key(stored_key, provided_password):

    salt = stored_key[:16]

    key = stored_key[16:]

    new_key =hashlib.pbkdf2_hmac('sha256', provided_password.encode('utf-8'), salt,
100000)

    return new_key == key



password = "secure_password"

key = generate_key(password)

print("Key generated successfully.")

print("Verification:", verify_key(key, password))
```

Presentación:

Parte 1: Prepara una presentación para resolver los anteriores problemas:

Estructura de la exposición:

1. Introducción:

- Saludo y presentación del presentador/es.
- Breve descripción del objetivo de la exposición y los temas que se cubrirán.
- Importancia de entender los conceptos de redes de computadoras en el contexto actual de la tecnología de la información.

2. Desarrollo de los temas:

- Fundamentos de Redes: Explicación de los conceptos básicos como ARPANET, backbone, Bluetooth, broadcast, cache memory, checksum, etc.
- Seguridad y fiabilidad de redes: Discusión sobre temas como encriptación, firewalls, TCP/IP, etc.
- Infraestructura de redes: Descripción de los componentes de una red, topologías, direccionamiento MAC, dispositivos de red, etc.
- Protocolos y comunicación: Explicación de protocolos como Ethernet, TCP/IP, OSI, así como los diferentes tipos de redes y protocolos de enrutamiento.

3. Estudios de caso:

- Análisis de estudios de caso reales relacionados con problemas de redes de computadoras y cómo se resolvieron.
- Discusión sobre lecciones aprendidas y mejores prácticas identificadas en cada caso.

4. Preguntas y respuestas:

- Sesión interactiva donde los asistentes pueden hacer preguntas sobre los temas tratados.
- Responder preguntas y proporcionar aclaraciones adicionales sobre los conceptos presentados.

6. Conclusión:

- Resumen de los puntos clave cubiertos durante la exposición.
- Reflexión sobre la importancia de comprender los conceptos de redes de computadoras en la era digital.
- Agradecimiento a los asistentes y cierre de la exposición.

Consejos para una exposición exitosa:

1. Conocimiento profundo: Asegúrate de comprender completamente los temas que vas a presentar y estar preparado para responder preguntas.

2. Claridad y concisión: Explica los conceptos de manera clara y sencilla, evitando tecnicismos innecesarios.
3. Ejemplos prácticos: Utiliza ejemplos prácticos y casos de uso para ilustrar los conceptos teóricos.
4. Interacción con la audiencia: Fomenta la participación de la audiencia haciendo preguntas o invitándolos a compartir sus experiencias.
5. Material visual atractivo: Utiliza diapositivas, gráficos o demostraciones visuales para mantener el interés de la audiencia.

Parte 2: Entrega de la actividad completa

Elementos a entregar:

1. Código fuente de la implementación del protocolo de red en un repositorio público en GitHub.
2. Presentación detallada en formato PDF, que incluya los objetivos, la metodología, los resultados y las conclusiones de la actividad y la presentación corregida.
3. En la presentación y en el repositorio, indicar claramente que se ha trabajado en equipo en el desarrollo del proyecto.

Fecha de entrega: 28 de abril hasta las 23:59

La actividad completa debe ser entregada en los repositorios individuales de los estudiantes en la fecha indicada.