actividad-13

April 30, 2024

1 PROBLEMA 1

Contexto: Una empresa necesita diseñar un sistema de correo electrónico robusto que utilice SMTP, IMAP, y SSL/TLS para la entrega y recuperación segura de correo electrónico.

[ ]:
```
import smtplib
from email.mime.text import MIMEText
import imaplib
import ssl
```

1.1 Paso 1: Configuración del servidor SMTP y IMAP en Python

[ ]:
```
# 3l siguiente código está mejorado en base al ejemplo brindado de la actividad
```

[ ]:
```
def setup_smtp_server(smtp_port=465): ## se usa este puerto cuando vamos a␣
 ↪enviar correos
    try:
        context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
        context.load_cert_chain(certfile="certificado.pem", keyfile="clave.pem")
        server = smtplib.SMTP_SSL('smtp.gmail.com', smtp_port, context=context)
        server.login('user@example.com', 'password') # ingresa el correo yc␣
 ↪contraseña desde donde enviarias un e-mail
        return server
    except Exception as e:
        print("Error en la conexión SMTP:", e)
        return None

def send_email(server):
    try:
        msg = MIMEText("Este es un correo electrónico automático de prueba.")
        msg['Subject'] = "Correo electrónico automático"
        msg['From'] = 'user@example.com' # ingresa el correo desde donde␣
 ↪enviarias un e-mail
        msg['To'] = 'recipient@example.com' # ingresa el correo de destino
        server.send_message(msg)
        print("Correo electrónico enviado")
    except Exception as e:
```

```
        print("Error al enviar el correo:", e)
    finally:
        if server:
            server.quit()

def setup_imap_server(imap_port=993):
    try:
        context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
        context.load_cert_chain(certfile="certificado.pem", keyfile="clave.pem")
        mail = imaplib.IMAP4_SSL('imap.gmail.com', imap_port, context=context)
        mail.login('user@example.com', 'password') # ingresa el correo yc␣
 ↪contraseña desde donde enviarias un e-mail
        return mail
    except Exception as e:
        print("Error en la conexión IMAP:", e)
        return None

def fetch_emails(mail, folder='inbox'):
    try:
        mail.select(folder)
        result, data = mail.search(None, 'ALL')
        mail_ids = data[0]
        id_list = mail_ids.split()
        latest_email_id = id_list[-1]
        result, data = mail.fetch(latest_email_id, '(RFC822)')
        raw_email = data[0][1]
        print(raw_email.decode('utf-8'))
    except Exception as e:
        print("Error al recuperar correos electrónicos:", e)
    finally:
        if mail:
            mail.logout()
```

[ ]:

1.2 Paso 2: Implementación de SSL/TLS

En la función `setup_smtp_server` se ha configurado una conexión **SMTP** segura con Gmail en el puerto 465. Establece una conexión usando `smtplib.SMTP_SSL`, inicia sesión con credenciales y devuelve el objeto del servidor si tiene éxito.

En la función `setup_imap_server` se configurado una conexión **IMAP** segura con Gmail. Utiliza `imaplib.IMAP4_SSL` para conectarse, inicia sesión con credenciales y devuelve el objeto de conexión si tiene éxito.

1.3 Paso 3: Manejo de Certificados X.509

```
[ ]: smtp_server = setup_smtp_server()
if smtp_server:
    send_email(smtp_server)
    smtp_server.quit()
    print("Correo electrónico automático enviado")
```

1.4 Paso 4: Discusión sobre DHCP y NAT

La asignación de direcciones IP dinámicas a través de **DHCP** y la traducción de direcciones **IP** realizada por **NAT** pueden complicar el acceso a los servidores de correo. Para solucionar este problema, se pueden implementar soluciones como la configuración de **NAT estático**, que asigna una dirección **IP pública fija** al servidor de correo, o el uso de servicios **DNS dinámicos** que asocien un nombre de dominio al servidor y actualicen automáticamente su dirección **IP**, facilitando así el acceso desde el exterior de la red local.

```
[ ]:
```

2 PROBLEMA 2

Contexto: Diseñar un protocolo de aplicación personalizado para un sistema de archivos distribuido que se ejecutará sobre TCP, utilizando técnicas como multiplexación y control de flujo.

```
[ ]: import socket
import struct
```

2.1 Paso 1: Diseño del protocolo

EL protocolo van a tener las siguientes operaciones básicas sobre el TCP:

PUT: Enviar un archivo al sistema.

GET: Recuperar un archivo del sistema.

DELETE: Eliminar un archivo del sistema.

Cada mensaje tendrá una cabecera que incluye el tipo de operación, el tamaño del mensaje, y un número de secuencia para el control de flujo y la recuperación de errores.

2.2 Paso 2: Implementación de control de flujo

```
[ ]: import socket
import struct
def send_message(sock, msg_type, seq_num, data):
    header = struct.pack('!I I', msg_type, seq_num)
    message = header + data.encode()
    sock.sendall(message)
def receive_message(sock):
```

```
    header = sock.recv(8)
    msg_type, seq_num = struct.unpack('!I I', header)
    data = sock.recv(1024) # ajustar según el tamaño esperado del mensaje
    return msg_type, seq_num, data.decode()

def main():
    host = 'localhost'
    port = 9000
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((host, port))
    server.listen(1)
    print("Server listening on port", port)
    client_sock, addr = server.accept()
    print("Connected by", addr)
    # Simulación de recepción de un mensaje
    msg_type, seq_num, data = receive_message(client_sock)
    print("Received:", msg_type, seq_num, data)
    # Envío de una respuesta
    send_message(client_sock, 1, seq_num + 1, "Ack")
    client_sock.close()
    server.close()

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

El código anterior funciona de la siguiente manera para una sola ves, cuando un cliente se conecta, el servidor espera recibir un mensaje con un formato específico, lo procesa y luego envía una respuesta de confirmación al cliente. Es decir, es la una estructura **TCP** para poder aplicar lo que nos pide el probelma.

Implementación de protocolo de aplicación personalizado para un sistema de archivos distribuido

[ ]:
```
import socket
import struct

# Definir las operaciones PUT; GET y DELETE para acceder
OPERATION_PUT = 1
OPERATION_GET = 2
OPERATION_DELETE = 3

def send_message(sock, operation, seq_num, data):
    header = struct.pack('!I I I', operation, len(data), seq_num)
    message = header + data.encode()
    sock.sendall(message)

def receive_message(sock):
    header = sock.recv(12)
```

```
    operation, msg_length, seq_num = struct.unpack('!I I I', header)
    data = sock.recv(msg_length).decode()
    return operation, seq_num, data

def put(sock, filename, seq_num): ## crear nuevos archivos
    with open(filename, 'rb') as f:
        file_data = f.read()
    send_message(sock, OPERATION_PUT, seq_num, file_data)

def get(sock, filename, seq_num): ## obtener algún archivo
    send_message(sock, OPERATION_GET, seq_num, filename)

    # Recibir el contenido del archivo
    operation, _, data = receive_message(sock)
    if operation == OPERATION_GET:
        with open(filename, 'wb') as f:
            f.write(data.encode())
        print(f"File '{filename}' received successfully.")
    else:
        print("Error: File not received.")

def delete(sock, filename, seq_num): ## eliminar algún archivo
    send_message(sock, OPERATION_DELETE, seq_num, filename)
    print(f"File '{filename}' deleted successfully.")

def main():
    host = 'localhost'
    port = 1245
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((host, port))
    server.listen(1)
    print("Server listening on port", port)
    client_sock, addr = server.accept()
    print("Connected by", addr)

    # Respecto a lo que se desea podemos usar las operaciones del problema
    seq_num = 1
    put(client_sock, "file.txt", seq_num)
    seq_num += 1
    get(client_sock, "file.txt", seq_num)
    seq_num += 1
    delete(client_sock, "file.txt", seq_num)

    client_sock.close()
    server.close()

if __name__ == '__main__':
```

```
    main()
```

1.- Las funciones `put, get, y delete` son ahora funciones separadas que toman un socket, el nombre del archivo y un número de secuencia como argumentos.

2.- La función `put` lee el contenido del archivo especificado, lo envía al servidor como un mensaje con la operación **OPERATION_PUT**.

3.- La función *get* envía una solicitud al servidor para obtener un archivo, luego recibe el contenido del archivo del servidor y lo guarda localmente.

4.- La función `delete` envía una solicitud al servidor para eliminar un archivo.

5.- En la función `main`, se ilustra un ejemplo de cómo usar estas funciones en secuencia para realizar operaciones `PUT, GET y DELETE` en archivos.

2.3 Paso 3: Evaluación del protocolo

Se recomienda el uso de la herramienta Wireshark para monitorear la eficacia del control de flujo y el manejo de errores durante la transferencia de archivos. Esto podría involucrar la simulación de condiciones de red adversas, como alta latencia y pérdida de paquetes, para ver cómo el protocolo se comporta y se recupera de estos problemas.

3 PROBLEAM 3

Contexto: Una organización requiere un sistema de autenticación segura que utilice LDAP para la gestión de identidades y SSH para el acceso remoto.

3.1 Paso 1: Configuración de LDAP y SSH

[ ]:
```
!pip install paramiko
!pip install ldap3
```

```
Requirement already satisfied: paramiko in /usr/local/lib/python3.10/dist-
packages (3.4.0)
Requirement already satisfied: bcrypt>=3.2 in /usr/local/lib/python3.10/dist-
packages (from paramiko) (4.1.2)
Requirement already satisfied: cryptography>=3.3 in
/usr/local/lib/python3.10/dist-packages (from paramiko) (42.0.5)
Requirement already satisfied: pynacl>=1.5 in /usr/local/lib/python3.10/dist-
packages (from paramiko) (1.5.0)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-
packages (from cryptography>=3.3->paramiko) (1.16.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-
packages (from cffi>=1.12->cryptography>=3.3->paramiko) (2.22)
Collecting ldap3
  Downloading ldap3-2.9.1-py2.py3-none-any.whl (432 kB)
                           432.2/432.2
kB 7.5 MB/s eta 0:00:00
```

```
Requirement already satisfied: pyasn1>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from ldap3) (0.6.0)
Installing collected packages: ldap3
Successfully installed ldap3-2.9.1
```

[ ]:
```python
import ldap3
import paramiko

def create_ssh_tunnel(user, password, host, remote_host, local_port,␣
 ↪remote_port):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(host, username=user, password=password)

    # Establecer un reenvío de puertos para LDAP
    tunnel = client.get_transport().open_channel('direct-tcpip', (remote_host,␣
 ↪remote_port), ('localhost', local_port))
    return client, tunnel


def main():
    # Configuración SSH
    ssh_user = 'admin'
    ssh_password = 'securepassword'
    ssh_host = 'example.com'

    # Configuración LDAP
    ldap_host = 'ldap.example.com'
    ldap_port = 389
    ldap_base_dn = 'dc=example,dc=com'
    ldap_search_filter = '(objectclass=person)'

    # Puerto local para el túnel SSH
    local_ldap_port = 389
    # Puerto remoto para el servidor LDAP
    remote_ldap_port = 389

    # Establecer túnel SSH
    client, tunnel = create_ssh_tunnel(ssh_user, ssh_password, ssh_host,␣
 ↪ldap_host, local_ldap_port, remote_ldap_port)
    print(f"SSH tunnel established for LDAP on port {local_ldap_port}")

    # Configurar conexión LDAP a través del túnel SSH
    ldap_server = ldap_host + ':' + str(local_ldap_port)
    ldap_conn = ldap3.Connection(ldap_server, auto_bind=True,␣
 ↪client_strategy=ldap3.SYNC)
```

```
    # Realizar operaciones LDAP a través del túnel SSH
    try:
        ldap_conn.search(ldap_base_dn, ldap_search_filter, ldap3.SUBTREE)
        print("LDAP search successful:")
        for entry in ldap_conn.entries:
            print(entry)
    except ldap3.LDAPException as e:
        print(f"LDAP search failed: {e}")

    # Cerrar túnel SSH y conexión LDAP
    tunnel.close()
    client.close()
    ldap_conn.unbind()


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

3.2 Paso 2: Implementación de seguridad

[ ]:
```
import ldap3
import paramiko
import ssl
from OpenSSL import crypto

def create_ssh_tunnel(user, password, host, remote_host, local_port,␣
 ↪remote_port):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(host, username=user, password=password)

    # Configurar SSL/TLS para el túnel SSH
    tunnel = client.get_transport().open_channel('direct-tcpip', (remote_host,␣
 ↪remote_port), ('localhost', local_port))

    return client, tunnel

def main():
    # Configuración SSH
    ssh_user = 'admin'
    ssh_password = 'securepassword'
    ssh_host = 'example.com'

    # Configuración LDAP
    ldap_host = 'ldap.example.com'
    ldap_port = 389
```

```
    ldap_base_dn = 'dc=example,dc=com'
    ldap_search_filter = '(objectclass=person)'

    # Puerto local para el túnel SSH
    local_ldap_port = 389
    # Puerto remoto para el servidor LDAP
    remote_ldap_port = 389

    # Establecer túnel SSH
    client, tunnel = create_ssh_tunnel(ssh_user, ssh_password, ssh_host,␣
 ↪ldap_host, local_ldap_port, remote_ldap_port)
    print(f"SSH tunnel established for LDAP on port {local_ldap_port}")

    # Configurar conexión LDAP a través del túnel SSH con SSL/TLS
    ldap_server = ldap_host + ':' + str(local_ldap_port)
    ldap_conn = ldap3.Connection(ldap_server, auto_bind=True,␣
 ↪client_strategy=ldap3.SYNC, use_ssl=True, ssl_version=ssl.PROTOCOL_TLSv1_2)

    # Realizar operaciones LDAP a través del túnel SSH
    try:
        ldap_conn.search(ldap_base_dn, ldap_search_filter, ldap3.SUBTREE)
        print("LDAP search successful:")
        for entry in ldap_conn.entries:
            print(entry)
    except ldap3.LDAPException as e:
        print(f"LDAP search failed: {e}")

    # Cerrar túnel SSH y conexión LDAP
    tunnel.close()
    client.close()
    ldap_conn.unbind()

if __name__ == '__main__':
    main()

def create_self_signed_cert(cert_file, key_file):
    k = crypto.PKey()
    k.generate_key(crypto.TYPE_RSA, 2048)
    cert = crypto.X509()
    cert.get_subject().C = "US"
    cert.get_subject().ST = "California"
    cert.get_subject().L = "San Francisco"
    cert.get_subject().O = "My Company"
    cert.get_subject().OU = "My Organizational Unit"
    cert.get_subject().CN = "mydomain.com"
    cert.set_serial_number(1000)
    cert.gmtime_adj_notBefore(0)
```

```
    cert.gmtime_adj_notAfter(10 * 365 * 24 * 60 * 60)
    cert.set_issuer(cert.get_subject())
    cert.set_pubkey(k)
    cert.sign(k, 'sha256')
    open(cert_file, "wt").write(crypto.dump_certificate(crypto.FILETYPE_PEM,␣
↪cert).decode('utf-8'))
    open(key_file, "wt").write(crypto.dump_privatekey(crypto.FILETYPE_PEM, k).
↪decode('utf-8'))

create_self_signed_cert('ldap_cert.pem', 'ldap_key.pem')
```

3.3 Paso 3: Evaluación de seguridad

Después de analizar el código se puede realizar algunas mejoras a partir de los conocimientos implementados en clase:

1. Identificar posibles amenazas como ataques de intermediario y configuraciones erróneas de certificados que podrían comprometer la seguridad de la comunicación **LDAP** a través del túnel **SSH**. A lvez verificar los certificados SSL/TLS utilizados para prevenir la suplantación de identidad del servidor.
2. Realizar pruebas para comprender posibles vulnerabilidades y puntos débiles en el sistema.
3. Monitorear las configuraciones de **SSH** y **LDAP** para garantizar que estén correctamente implementadas y sigan las mejores prácticas de seguridad.
4. Implementar sistemas de monitorización y registro para detectar actividades sospechosas o intentos de acceso no autorizado y responder rápidamente a incidentes de seguridad.

4 PROBLEMA 4

Contexto: Simular un entorno de red que utilice múltiples protocolos de la pila TCP/IP, asegurando la interoperabilidad entre dispositivos que utilizan diferentes configuraciones de red.

4.1 Paso 1: Simulación de protocolos de red en Python

[ ]:
```
from scapy.all import *
def simulate_ip():
    packet = IP(dst="192.168.1.1") / ICMP() / "Hello, this is an IP packet"
    send(packet)
def simulate_icmp():
    icmp_echo = IP(dst="192.168.1.1") / ICMP(type=8, code=0) / "Ping"
    send(icmp_echo)

def simulate_igmp():
    igmp_packet = IP(dst="224.0.0.1") / IGMP(type=0x16, gaddr="224.0.0.1")
    send(igmp_packet)
def simulate_arp():
    arp_request = ARP(pdst='192.168.1.2')
```

```
    send(arp_request)

simulate_ip()
simulate_icmp()
simulate_igmp()
simulate_arp()
```

4.2 Paso 2: Evaluación de interoperabilidad

Para evaluar la interoperabilidad, se puede utilizar **Wireshark** para analizar el tráfico de red después de simular cada protocolo, puedes obtener una visión detallada del rendimiento, la eficiencia y la seguridad de tu red. Esto te permite identificar y resolver problemas potenciales, optimizar el rendimiento y garantizar una comunicación fluida entre dispositivos.

4.3 Paso 3: Uso de R-utilities para diagnóstico

[ ]:
```python
import os
def run_traceroute(target):
    response = os.system(f"traceroute {target}")
    print(response)
def run_ping(target):
    response = os.system(f"ping -c 4 {target}")
    print(response)
run_traceroute('192.168.1.1')
run_ping('192.168.1.1')
```

```
32512
32512
```