Boletín de Ejercicios Prácticos

Redes de Computadores: Scripts y Programación

Introducción

Este boletín contiene ejercicios prácticos donde deberás completar scripts y programas relacionados con redes y protocolos. Los archivos a completar están disponibles en la carpeta de ejercicios.

Estructura de archivos:

- `ejercicio_01.bat` - Script Windows (Batch)
- `ejercicio_02.sh` - Script Linux (Bash)
- `ejercicio_03.py` - Programa Python
- `ejercicio_04.rs` - Programa Rust
- `ejercicio_05.bat` - Script Windows avanzado
- `ejercicio_06.sh` - Script Linux avanzado
- `ejercicio_07.py` - Cliente HTTP en Python
- `ejercicio_08.rs` - Servidor TCP en Rust

1. Ejercicio 1: Script de Diagnóstico de Red (Windows BAT)

Archivo: `ejercicio_01.bat`
Objetivo: Completar un script batch que realice diagnósticos básicos de red en Windows.
Tareas a completar:

1. Mostrar la configuración IP actual
2. Hacer ping al gateway predeterminado
3. Resolver el dominio www.google.com
4. Mostrar la tabla ARP
5. Mostrar las conexiones activas

Conceptos aplicados: Comandos ipconfig, ping, nslookup, arp, netstat

2. Ejercicio 2: Script de Configuración de Red (Linux Bash)

Archivo: `ejercicio_02.sh`
Objetivo: Completar un script bash que configure una interfaz de red en Linux.
Tareas a completar:

1. Verificar que el script se ejecuta como root
2. Mostrar interfaces de red disponibles
3. Asignar una IP estática a una interfaz
4. Configurar la puerta de enlace predeterminada
5. Verificar la conectividad

Conceptos aplicados: Comandos ip, route, permisos de usuario, variables bash

3. Ejercicio 3: Escáner de Red en Python

Archivo: `ejercicio_03.py`
Objetivo: Completar un programa que escanee una subred y detecte hosts activos.
Tareas a completar:

1. Implementar función para calcular el rango de IPs de una subred
2. Realizar ping a cada IP del rango
3. Intentar resolver el nombre DNS de cada host activo
4. Generar un informe con los resultados

Conceptos aplicados: Sockets, ICMP, DNS, cálculo de subredes, concurrencia

4. Ejercicio 4: Cliente TCP en Rust

Archivo: `ejercicio_04.rs`
Objetivo: Completar un cliente TCP que se conecte a un servidor y envíe mensajes.
Tareas a completar:

1. Establecer conexión TCP con un servidor
2. Implementar el three-way handshake (automático con TcpStream)
3. Enviar datos al servidor
4. Recibir y mostrar la respuesta
5. Manejar errores de conexión

Conceptos aplicados: TCP, manejo de streams, serialización, gestión de errores

5. Ejercicio 5: Monitor de Tráfico (Windows BAT)

Archivo: `ejercicio_05.bat`
Objetivo: Crear un script que monitorice el tráfico de red y genere estadísticas.
Tareas a completar:

1. Capturar estadísticas de red cada 5 segundos
2. Filtrar conexiones por protocolo (TCP/UDP)
3. Identificar las IPs con más conexiones
4. Guardar logs en un archivo

Conceptos aplicados: netstat avanzado, parsing de texto, loops en batch

6. Ejercicio 6: Servidor DHCP Simulado (Linux Bash)

Archivo: `ejercicio_06.sh`
Objetivo: Simular el proceso DHCP asignando IPs a dispositivos.
Tareas a completar:

1. Leer archivo con MACs de dispositivos
2. Asignar IPs del pool disponible
3. Implementar la secuencia DORA (Discover, Offer, Request, Acknowledge)
4. Guardar las asignaciones en un archivo de leases
5. Implementar renovación y liberación de IPs

Conceptos aplicados: DHCP, gestión de archivos, asociative arrays en bash

7. Ejercicio 7: Cliente HTTP/HTTPS en Python

Archivo: `ejercicio_07.py`
Objetivo: Implementar un cliente HTTP que realice peticiones y analice respuestas.
Tareas a completar:

1. Realizar petición GET a una URL
2. Implementar soporte para HTTPS (TLS/SSL)
3. Parsear las cabeceras HTTP
4. Verificar certificados SSL
5. Seguir redirecciones
6. Manejar cookies

Conceptos aplicados: HTTP, HTTPS, TLS/SSL, sockets, certificados digitales

8. Ejercicio 8: Servidor TCP Multi-cliente en Rust

Archivo: `ejercicio_08.rs`
Objetivo: Crear un servidor TCP que pueda manejar múltiples clientes simultáneamente.
Tareas a completar:

1. Crear un socket TCP en modo escucha
2. Aceptar conexiones de múltiples clientes
3. Implementar manejo concurrente (threads o async)
4. Implementar un protocolo simple de chat
5. Broadcast de mensajes a todos los clientes
6. Gestionar desconexiones

Conceptos aplicados: TCP, concurrencia, threads, manejo de estado compartido

9. Ejercicio 9: Analizador de Paquetes ARP (Python)

Archivo: `ejercicio_09.py`
Objetivo: Capturar y analizar tráfico ARP para detectar posibles ataques.
Tareas a completar:

1. Capturar paquetes ARP en la red local
2. Parsear los campos del protocolo ARP
3. Detectar ARP spoofing (misma IP con diferente MAC)
4. Mantener una tabla ARP local
5. Generar alertas ante comportamientos sospechosos

Conceptos aplicados: Protocolo ARP, raw sockets, sniffing, seguridad

10. Ejercicio 10: Resolución DNS en Rust

Archivo: `ejercicio_10.rs`
Objetivo: Implementar un cliente DNS que resuelva nombres de dominio.
Tareas a completar:

1. Construir una petición DNS en formato binario
2. Enviar la petición a un servidor DNS (UDP puerto 53)
3. Parsear la respuesta DNS
4. Extraer las direcciones IP del resultado
5. Implementar caché local de DNS
6. Soportar diferentes tipos de registros (A, AAAA, MX, NS)

Conceptos aplicados: DNS, UDP, formato binario, parsing de protocolos

11. Ejercicio 11: Escáner de Puertos (Python)

Archivo: `ejercicio_11.py`
Objetivo: Crear un escáner de puertos que identifique servicios activos.
Tareas a completar:

1. Escanear un rango de puertos de una IP
2. Identificar puertos abiertos, cerrados y filtrados
3. Determinar el servicio que corre en cada puerto
4. Implementar diferentes técnicas de escaneo (TCP connect, SYN scan)
5. Añadir capacidad de escaneo concurrente
6. Generar informe de servicios detectados

Conceptos aplicados: TCP, sockets, puertos, servicios, concurrencia

12. Ejercicio 12: Proxy HTTP en Rust

Archivo: `ejercicio_12.rs`
Objetivo: Implementar un proxy HTTP básico que intercepte y reenvíe peticiones.
Tareas a completar:

1. Escuchar conexiones de clientes HTTP
2. Parsear peticiones HTTP
3. Establecer conexión con el servidor destino
4. Reenviar peticiones y respuestas
5. Implementar caché de respuestas
6. Añadir logging de todas las peticiones

Conceptos aplicados: HTTP, proxy, TCP, parsing, caché

Recursos Adicionales

- RFC 791 (IP), RFC 792 (ICMP), RFC 793 (TCP)
- RFC 826 (ARP), RFC 1035 (DNS), RFC 2131 (DHCP)
- Documentación de Python: `https://docs.python.org/3/library/socket.html`
- The Rust Book: `https://doc.rust-lang.org/book/`
- Wireshark para análisis de tráfico