UNIVERSIDAD DEL ISTMO
FACULTAD DE INGENIERÍA

PROYECTO 2
Spoofing

CARLOS SOLARES

Catedrático: Juan Miguel Lopez Velasquez

Guatemala, 18 de Septiembre del 2024

1. Introducción

Descripción del Proyecto: Aquí se describe el propósito del proyecto, que es crear y analizar paquetes TCP/IP manualmente. Incluye los objetivos de la investigación, como entender el formato de los paquetes y las técnicas de validación de checksum.

Objetivos:

- Crear paquetes TCP/IP manualmente en IPv4 e IPv6.
- Validar los paquetes usando herramientas como Wireshark.
- Documentar el proceso y los desafíos encontrados.

2. Fundamentos Teóricos

2.1. Protocolo TCP/IP

TCP (Transmission Control Protocol):

- **Descripción General**: TCP es un protocolo de transporte orientado a la conexión que garantiza la entrega fiable y ordenada de datos entre aplicaciones. Establece una conexión entre el origen y el destino antes de transmitir datos y asegura que los paquetes lleguen sin errores y en el orden correcto.
- **Campos Clave en el Encabezado TCP**:
    - **Puertos de Origen y Destino**: Los puertos especifican las aplicaciones en cada extremo de la conexión. Un puerto es un número de 16 bits.
    - **Número de Secuencia**: Un número de 32 bits que identifica el orden de los bytes enviados en la conexión. Es esencial para la reordenación y la recuperación de datos.
    - **Número de Acuse de Recibo**: Un número de 32 bits que indica el próximo byte esperado por el receptor. Utilizado para confirmar la recepción de datos.
    - **Flags**: Campos de 1 bit que controlan el estado de la conexión. Ejemplos incluyen SYN (inicialización de conexión), ACK (confirmación), FIN (finalización de conexión).
    - **Ventana**: Campo de 16 bits que especifica la cantidad de datos que el receptor está dispuesto a recibir.
    - **Checksum**: Campo de 16 bits utilizado para verificar la integridad de los datos en el encabezado y la carga útil. El cálculo del checksum asegura que los datos no se corrompan durante la transmisión.
    - **Puntero Urgente**: Campo de 16 bits que indica el final de los datos urgentes, si se utiliza.

IP (Internet Protocol):

- **IPv4 (Internet Protocol versión 4)**:

    - **Descripción General**: IPv4 es el protocolo de red que utiliza direcciones de 32 bits, proporcionando alrededor de 4.3 mil millones de direcciones únicas. Es la versión más antigua y ampliamente utilizada del protocolo IP.
    - **Campos Clave en el Encabezado IPv4**:
        - **Versión**: Campo de 4 bits que indica la versión del protocolo (para IPv4, el valor es 4).
        - **Longitud del Encabezado**: Campo de 4 bits que indica la longitud del encabezado IP.
        - **Tipo de Servicio**: Campo de 8 bits que se utiliza para la calidad del servicio (QoS), incluyendo prioridad y control de congestión.
        - **Longitud Total**: Campo de 16 bits que indica la longitud total del datagrama IP, incluidos los encabezados y la carga útil.
        - **Identificación**: Campo de 16 bits utilizado para identificar fragmentos de un datagrama IP.
        - **Fragmentación**: Campos que manejan la fragmentación y reensamblaje de datagramas.
        - **TTL (Time to Live)**: Campo de 8 bits que limita la vida útil del paquete para evitar que circule indefinidamente.
        - **Protocolo**: Campo de 8 bits que especifica el protocolo de nivel superior (TCP, UDP, etc.).
        - **Checksum del Encabezado**: Campo de 16 bits utilizado para verificar la integridad del encabezado IP.
        - **Direcciones IP de Origen y Destino**: Campos de 32 bits que especifican la dirección de origen y destino del paquete.
- **IPv6 (Internet Protocol versión 6)**:
    - **Descripción General**: IPv6 es la versión más reciente del protocolo IP, que utiliza direcciones de 128 bits, proporcionando un espacio de direcciones mucho más grande que IPv4. También incluye mejoras en el encabezado para simplificar el procesamiento de paquetes.
    - **Campos Clave en el Encabezado IPv6**:
        - **Versión**: Campo de 4 bits que indica la versión del protocolo (para IPv6, el valor es 6).
        - **Clase de Tráfico**: Campo de 8 bits que se utiliza para la calidad del servicio.
        - **Etiqueta de Flujo**: Campo de 20 bits que identifica flujos de datos especiales.
        - **Longitud del Payload**: Campo de 16 bits que indica la longitud de la carga útil del paquete, excluyendo el encabezado.
        - **Próximo Encabezado**: Campo de 8 bits que indica el protocolo de nivel superior (TCP, UDP, etc.).
        - **Límite de Salto**: Campo de 8 bits que limita el número de saltos que puede hacer el paquete.
        - **Direcciones IP de Origen y Destino**: Campos de 128 bits que especifican las direcciones de origen y destino del paquete.

2.2. Construcción de Paquetes TCP/IP

Encabezado IPv4:

- **Versión**: 4 bits que indican la versión del protocolo IP (IPv4 es 4).
- **Longitud del Encabezado**: 4 bits que especifican la longitud del encabezado IP, en palabras de 32 bits.
- **Tipo de Servicio**: 8 bits que proporcionan información sobre el servicio deseado para el paquete.
- **Longitud Total**: 16 bits que indican la longitud total del datagrama, incluyendo los encabezados y la carga útil.
- **Identificación**: 16 bits utilizados para identificar fragmentos de un datagrama.
- **Fragmentación**: Campos que manejan la fragmentación del datagrama IP.
- **TTL (Time to Live)**: 8 bits que evitan que los paquetes se envíen indefinidamente.
- **Protocolo**: 8 bits que especifican el protocolo de capa superior que usa el paquete (TCP, UDP, etc.).
- **Checksum del Encabezado**: 16 bits utilizados para verificar la integridad del encabezado IP.
- **Direcciones IP de Origen y Destino**: 32 bits para cada dirección IP.

Encabezado IPv6:

- **Versión**: 4 bits que indican que el protocolo es IPv6.
- **Clase de Tráfico**: 8 bits que se utilizan para priorizar los paquetes.
- **Etiqueta de Flujo**: 20 bits que identifican flujos de datos específicos.
- **Longitud del Payload**: 16 bits que indican la longitud de la carga útil.
- **Próximo Encabezado**: 8 bits que especifican el protocolo de capa superior.
- **Límite de Salto**: 8 bits que limitan el número de saltos permitidos para el paquete.
- **Direcciones IP de Origen y Destino**: 128 bits para cada dirección IP.

Checksum TCP/IP:

- **Método de Cálculo del Checksum**: El checksum se calcula sumando todos los 16 bits en el encabezado y la carga útil y luego tomando el complemento a uno de la suma. Para TCP/IP, se utiliza un pseudo-encabezado para asegurar la integridad del encabezado y la carga útil.
- **Importancia del Pseudo-Encabezado**: El pseudo-encabezado incluye información del encabezado IP que es esencial para la validación del checksum. En IPv4, incluye las direcciones IP de origen y destino, el protocolo y la longitud del encabezado TCP. En IPv6, incluye las direcciones IP de origen y destino, el próximo encabezado y la longitud del TCP.
- **Validación del Checksum**: La verificación del checksum en el receptor asegura que los datos no hayan sido alterados durante la transmisión. Si el checksum calculado en el receptor no coincide con el valor incluido en el paquete, se considera que el paquete es corrupto y se descarta.

3. Código y Funcionalidad

3.1. Descripción General

El programa en C# tiene como objetivo principal construir y enviar paquetes TCP/IP personalizados. Los paquetes pueden ser IPv4 o IPv6, y el programa permite al usuario especificar las direcciones IP de origen y destino. La funcionalidad incluye la creación de los encabezados IP y TCP, el cálculo de los checksums necesarios y el envío de los paquetes a través de sockets.

3.2. Análisis del Código

3.2.1. Función `Main`

La función `Main` es el punto de entrada del programa:

```
static void Main(string[] args)
{
    string sourceIp = GetValidIP("origen", null);
    if (sourceIp == null) return; // Usuario eligió 'exit'

    string destIp = GetValidIP("destino", sourceIp);
    if (destIp == null) return; // Usuario eligió 'back' o 'exit'

    if (IPAddress.TryParse(sourceIp, out IPAddress srcAddress) &&
     IPAddress.TryParse(destIp, out IPAddress dstAddress))
    {
        if (srcAddress.AddressFamily == AddressFamily.InterNetwork)
        {
            Console.WriteLine("Creando paquete IPv4...");
            byte[] packet = CreateIPv4Packet(srcAddress, dstAddress);
            SendPacket(packet, AddressFamily.InterNetwork);
        }
        else if (srcAddress.AddressFamily == AddressFamily.InterNetworkV6)
        {
```

```
            Console.WriteLine("Creando paquete IPv6...");
            byte[] packet = CreateIPv6Packet(srcAddress, dstAddress);
            SendPacket(packet, AddressFamily.InterNetworkV6);
        }
    }
}
```

1. **Obtención de IPs**: Solicita al usuario las direcciones IP de origen y destino. Si el usuario elige salir (`exit`) o volver (`back`), el programa se cierra o regresa a la opción anterior, respectivamente.
2. **Validación de IPs**: Verifica si las IPs ingresadas son válidas y determina si son IPv4 o IPv6.
3. **Creación y Envío de Paquete**: Dependiendo del tipo de IP, se crea el paquete adecuado y se envía utilizando sockets.

3.2.2. Función `GetValidIP`

```
static string GetValidIP(string tipo, string previousIp)
{
    string ip;
    bool isValid = false;

    do
    {
        Console.WriteLine($"Ingrese la IP de {tipo} (escriba 'back' para regresar o 'exit' para salir):");
        ip = Console.ReadLine();

        if (ip.ToLower() == "exit")
        {
            Console.WriteLine("Saliendo del programa...");
```

```
            Environment.Exit(0);
        }
        else if (ip.ToLower() == "back" && previousIp != null)
        {
            Console.WriteLine("Regresando a la opción anterior...");
            return null;
        }

        isValid = IsValidIP(ip);

        if (!isValid)
        {
            Console.WriteLine($"La IP de {tipo} '{ip}' no es válida. Inténtalo de
    nuevo.");
        }
    } while (!isValid);

    return ip;
}
```

1. **Entrada de IP**: Solicita al usuario una dirección IP, permitiendo volver o salir.
2. **Validación**: Llama a `IsValidIP` para verificar la validez de la IP ingresada.
3. **Retorno de IP**: Devuelve la IP válida o maneja las opciones de salida y regreso.

3.2.3. Función `IsValidIP`

```
static bool IsValidIP(string ip)
{
    if (IPAddress.TryParse(ip, out IPAddress address))
```

```
        {
            if (address.AddressFamily == AddressFamily.InterNetwork)
            {
                Console.WriteLine($"{ip} es una dirección IPv4.");
                return true;
            }
            else if (address.AddressFamily == AddressFamily.InterNetworkV6)
            {
                Console.WriteLine($"{ip} es una dirección IPv6.");
                return true;
            }
        }

        return false;
    }
```

1. **Validación de IP**: Verifica si la dirección IP es válida y especifica si es IPv4 o IPv6.

3.2.4. Función `CreateIPv4Packet`

```
static byte[] CreateIPv4Packet(IPAddress sourceIp, IPAddress destIp)
{
    byte[] ipHeader = new byte[20];
    byte[] tcpHeader = new byte[20];
    byte[] packet = new byte[ipHeader.Length + tcpHeader.Length];

    // Construcción del Encabezado IPv4
```

```
ipHeader[0] = 0x45; // Version (4 bits) + IHL (4 bits)
ipHeader[1] = 0x00; // Tipo de Servicio
ipHeader[2] = 0x00; // Longitud Total (Se actualizará más adelante)
ipHeader[3] = 0x28; // Longitud Total = 40 bytes (20 IP + 20 TCP)

ipHeader[4] = 0x00; // Identificación
ipHeader[5] = 0x00;

ipHeader[6] = 0x40; // Flags (No fragmentar) + Fragment Offset
ipHeader[7] = 0x00;

ipHeader[8] = 64;    // TTL
ipHeader[9] = 6;     // Protocolo TCP

// Checksum IPv4 (inicialmente 0, se calculará después)
ipHeader[10] = 0x00;
ipHeader[11] = 0x00;

// Direcciones IP
Array.Copy(sourceIp.GetAddressBytes(), 0, ipHeader, 12, 4); // IP origen
Array.Copy(destIp.GetAddressBytes(), 0, ipHeader, 16, 4);   // IP destino

// Calcula el checksum IPv4
ushort ipChecksum = CalculateChecksum(ipHeader, ipHeader.Length);
ipHeader[10] = (byte)(ipChecksum >> 8);
```

```
ipHeader[11] = (byte)(ipChecksum & 0xFF);

// Construcción del Encabezado TCP
ushort sourcePort = 12345;
ushort destPort = 80;
tcpHeader[0] = (byte)(sourcePort >> 8);
tcpHeader[1] = (byte)(sourcePort & 0xFF);
tcpHeader[2] = (byte)(destPort >> 8);
tcpHeader[3] = (byte)(destPort & 0xFF);

// Número de secuencia
tcpHeader[4] = 0x00;
tcpHeader[5] = 0x00;
tcpHeader[6] = 0x00;
tcpHeader[7] = 0x01;

// Número de acuse de recibo
tcpHeader[8] = 0x00;
tcpHeader[9] = 0x00;
tcpHeader[10] = 0x00;
tcpHeader[11] = 0x00;

// Offset de datos (5) + Reservado + Flags (SYN)
tcpHeader[12] = 0x50; // Data offset = 5 (20 bytes), reservados
tcpHeader[13] = 0x02; // Flags: SYN
```

```
// Ventana
tcpHeader[14] = 0xFF;
tcpHeader[15] = 0xFF;

// Checksum TCP (inicialmente 0, se calculará después)
tcpHeader[16] = 0x00;
tcpHeader[17] = 0x00;

// Puntero urgente
tcpHeader[18] = 0x00;
tcpHeader[19] = 0x00;

// Construir el paquete completo (IP + TCP)
Array.Copy(ipHeader, 0, packet, 0, ipHeader.Length);
Array.Copy(tcpHeader, 0, packet, ipHeader.Length, tcpHeader.Length);

// Calcula el checksum TCP
ushort tcpChecksum = CalculateTcpChecksum(ipHeader, tcpHeader);
tcpHeader[16] = (byte)(tcpChecksum >> 8);
tcpHeader[17] = (byte)(tcpChecksum & 0xFF);

// Actualizar el checksum en el paquete
Array.Copy(tcpHeader, 0, packet, ipHeader.Length, tcpHeader.Length);
```

```
    return packet;
}
```

1. **Encabezado IPv4**:
    - **Version y IHL**: Define la versión IP y la longitud del encabezado.
    - **Tipo de Servicio**: Define la prioridad del paquete.
    - **Longitud Total**: La longitud total del paquete.
    - **Identificación, Flags, y Fragment Offset**: Controlan la fragmentación.
    - **TTL y Protocolo**: TTL es el tiempo de vida, y el protocolo es TCP.
    - **Checksum**: Verifica la integridad del encabezado IP.
2. **Encabezado TCP**:
    - **Puertos de Origen y Destino**: Define los puertos TCP.
    - **Número de Secuencia y Acuse de Recibo**: Controlan el flujo de datos.
    - **Offset de Datos, Reservado, y Flags**: Define el tipo de paquete (SYN en este caso).
    - **Ventana y Checksum**: Controla el tamaño de la ventana y verifica la integridad del encabezado TCP.
3. **Construcción del Paquete**: Une los encabezados IP y TCP para formar el paquete completo.
4. **Cálculo del Checksum**:
    - **IPv4**: `CalculateChecksum` calcula el checksum del encabezado IP.
    - **TCP**: `CalculateTcpChecksum` calcula el checksum del encabezado TCP, usando el encabezado IP para obtener la suma de verificación adecuada.

3.2.5. Función `CreateIPv6Packet`

```
static byte[] CreateIPv6Packet(IPAddress sourceIp, IPAddress destIp)
{
    byte[] ipHeader = new byte[40];
    byte[] tcpHeader = new byte[20];
    byte[] packet = new byte[ipHeader.Length + tcpHeader.Length];

    // Construcción del Encabezado IPv6
    ipHeader[0] = 0x60; // Versión (6) + Prioridad (0)
    ipHeader[1] = 0x00; // Prioridad
```

```
ipHeader[2] = 0x00; // Longitud de Carga Útil (se actualizará más tarde)
ipHeader[3] = 0x28;

// Identificador de Flujo
ipHeader[4] = 0x00;
ipHeader[5] = 0x00;
ipHeader[6] = 0x00;
ipHeader[7] = 0x00;

// Carga Útil (se actualizará más tarde)
ipHeader[8] = 0x00;
ipHeader[9] = 0x00;

// TTL
ipHeader[10] = 0x40; // TTL (64)

// Protocolo (TCP)
ipHeader[11] = 0x06; // TCP

// Direcciones IP
Array.Copy(sourceIp.GetAddressBytes(), 0, ipHeader, 8, 16); // IP origen
Array.Copy(destIp.GetAddressBytes(), 0, ipHeader, 24, 16); // IP destino

// Construcción del Encabezado TCP (igual que en IPv4)
ushort sourcePort = 12345;
```

```
ushort destPort = 80;
tcpHeader[0] = (byte)(sourcePort >> 8);
tcpHeader[1] = (byte)(sourcePort & 0xFF);
tcpHeader[2] = (byte)(destPort >> 8);
tcpHeader[3] = (byte)(destPort & 0xFF);

// Número de secuencia
tcpHeader[4] = 0x00;
tcpHeader[5] = 0x00;
tcpHeader[6] = 0x00;
tcpHeader[7] = 0x01;

// Número de acuse de recibo
tcpHeader[8] = 0x00;
tcpHeader[9] = 0x00;
tcpHeader[10] = 0x00;
tcpHeader[11] = 0x00;

// Offset de datos (5) + Reservado + Flags (SYN)
tcpHeader[12] = 0x50; // Data offset = 5 (20 bytes), reservados
tcpHeader[13] = 0x02; // Flags: SYN

// Ventana
tcpHeader[14] = 0xFF;
tcpHeader[15] = 0xFF;
```

```
    // Checksum TCP (inicialmente 0, se calculará después)
    tcpHeader[16] = 0x00;
    tcpHeader[17] = 0x00;

    // Puntero urgente
    tcpHeader[18] = 0x00;
    tcpHeader[19] = 0x00;

    // Construir el paquete completo (IP + TCP)
    Array.Copy(ipHeader, 0, packet, 0, ipHeader.Length);
    Array.Copy(tcpHeader, 0, packet, ipHeader.Length, tcpHeader.Length);

    // Calcula el checksum TCP
    ushort tcpChecksum = CalculateTcpChecksum(ipHeader, tcpHeader);
    tcpHeader[16] = (byte)(tcpChecksum >> 8);
    tcpHeader[17] = (byte)(tcpChecksum & 0xFF);

    // Actualizar el checksum en el paquete
    Array.Copy(tcpHeader, 0, packet, ipHeader.Length, tcpHeader.Length);
    return packet;
}
```

1. **Encabezado IPv6**:
    - **Versión y Prioridad**: Define la versión IP y la prioridad del paquete.
    - **Longitud de Carga Útil**: La longitud de los datos del paquete.
    - **Identificador de Flujo**: Identifica el flujo de datos.
    - **TTL y Protocolo**: TTL es el tiempo de vida y el protocolo es TCP.

2. **Encabezado TCP**: Similar al de IPv4.
3. **Construcción del Paquete**: Une los encabezados IP y TCP para formar el paquete completo.
4. **Cálculo del Checksum TCP**: `CalculateTcpChecksum` se utiliza para verificar la integridad del encabezado TCP.

4.2.6. Función `SendPacket`

```
static void SendPacket(byte[] packet, AddressFamily addressFamily)

{

    using (Socket socket = new Socket(addressFamily, SocketType.Raw,
     ProtocolType.Raw))

    {

        socket.SendTo(packet, new IPEndPoint(IPAddress.Any, 0));

        Console.WriteLine("Paquete enviado con éxito.");

    }

}
```

1. **Creación del Socket**: Crea un socket para el protocolo especificado (IPv4 o IPv6).
2. **Envío del Paquete**: Envía el paquete utilizando el socket.
3. **Mensaje de Confirmación**: Imprime un mensaje de éxito.

3.2.7. Función `CalculateChecksum`

```
static ushort CalculateChecksum(byte[] data, int length)

{

    uint sum = 0;

    for (int i = 0; i < length; i += 2)

    {

        sum += (uint)((data[i] << 8) + (i + 1 < length ? data[i + 1] : 0));

    }

    while (sum >> 16 > 0)

    {
```

```
            sum = (sum & 0xFFFF) + (sum >> 16);
        }
        return (ushort)~sum;
    }
```

1. **Cálculo del Checksum**: Suma los valores de 16 bits y ajusta el resultado para obtener el checksum.

3.2.8. Función `CalculateTcpChecksum`

```
static ushort CalculateTcpChecksum(byte[] ipHeader, byte[] tcpHeader)
{
    byte[] pseudoHeader = new byte[40];
    Array.Copy(ipHeader, 12, pseudoHeader, 0, 8); // IP Origen
    Array.Copy(ipHeader, 16, pseudoHeader, 8, 8); // IP Destino
    pseudoHeader[16] = 0; // Relleno
    pseudoHeader[17] = 6; // Protocolo (TCP)
    ushort tcpLength = (ushort)(tcpHeader.Length + 20); // Longitud TCP + Pseudoheader
    pseudoHeader[18] = (byte)(tcpLength >> 8);
    pseudoHeader[19] = (byte)(tcpLength & 0xFF);

    byte[] fullPacket = new byte[pseudoHeader.Length + tcpHeader.Length];
    Array.Copy(pseudoHeader, 0, fullPacket, 0, pseudoHeader.Length);
    Array.Copy(tcpHeader, 0, fullPacket, pseudoHeader.Length, tcpHeader.Length);

    return CalculateChecksum(fullPacket, fullPacket.Length);
}
```

1. **Encabezado Pseudo**: Construye el pseudo encabezado necesario para el cálculo del checksum TCP.
2. **Construcción del Paquete Completo**: Une el pseudo encabezado y TCP para calcular el checksum.

4. Bibliografía

Kurose, J. F., & Ross, K. W. (2017). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.

Comer, D. E. (2018). *Computer Networks and Internets* (6th ed.). Pearson.

Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley.

Microsoft. (n.d.). *System.Net.Sockets Namespace*. Retrieved from https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets

Microsoft. (n.d.). *IPAddress Class*. Retrieved from https://docs.microsoft.com/en-us/dotnet/api/system.net.ipaddress

Postel, J. (1981). *Internet Protocol*. RFC 791. Retrieved from https://tools.ietf.org/html/rfc791

Krawczyk, H., & Ellis, D. (1998). *IPsec: Internet Protocol Security*. RFC 2401. Retrieved from https://tools.ietf.org/html/rfc2401

Reynolds, J., & Postel, J. (1982). *File Transfer Protocol*. RFC 959. Retrieved from https://tools.ietf.org/html/rfc959

Balakrishnan, H., & D. H. K. (1999). *A Case for End-to-End Congestion Control in TCP/IP Networks*. IEEE Journal on Selected Areas in Communications, 17(2), 396-407.