UNIVERSIDAD DEL ISTMO
FACULTAD DE INGENIERÍA

PROYECTO
RIESGOS

CARLOS SOLARES

Guatemala, 17 de Abril del 2023

ICMP

El Protocolo de Mensajes de Control de Internet (ICMP, por sus siglas en inglés) es un protocolo de la capa de red utilizado para enviar mensajes de control y error entre dispositivos en una red IP. Algunas de las funciones comunes de ICMP incluyen la verificación de la conectividad de red, la resolución de problemas de enrutamiento y la generación de mensajes de error cuando ocurren problemas en la comunicación de red.

Tener ICMP habilitado puede tener varias repercusiones:

- Facilita la depuración de problemas de red: ICMP permite a los dispositivos de red enviar mensajes de error y control entre sí, lo que facilita la identificación y resolución de problemas de conectividad y enrutamiento en la red.
- Ping y rastreo de ruta: ICMP incluye comandos como ping y tracert que se utilizan comúnmente para probar la conectividad de red y diagnosticar problemas. Estas herramientas pueden proporcionar información valiosa sobre la latencia, la pérdida de paquetes y la ruta que sigue un paquete a través de la red.
- Potencial para ataques de denegación de servicio (DDoS): Algunos tipos de mensajes ICMP, como los paquetes de eco de solicitud de ping, pueden ser utilizados en ataques de inundación de ping. En estos ataques, se envían grandes cantidades de solicitudes de ping a un dispositivo o red, lo que puede saturar los recursos de red y causar una interrupción del servicio.
- Posible revelación de información sensible: Algunos dispositivos y sistemas pueden estar configurados para responder a los mensajes ICMP con información sensible, como nombres de host o direcciones IP internas. Esto puede representar un riesgo de seguridad si la información se revela a usuarios no autorizados.

En resumen, ICMP es una parte fundamental del funcionamiento de Internet y es útil para diagnosticar problemas de red, pero también puede ser utilizado en ataques maliciosos si no se controla adecuadamente su uso y su exposición en la red. Es importante configurar adecuadamente los dispositivos de red para equilibrar la utilidad de ICMP con la seguridad de la red.

PENTESTING

El pentesting, o prueba de penetración, es un proceso de evaluación de la seguridad de sistemas informáticos, redes o aplicaciones mediante la simulación de ataques de hackers éticos. El objetivo principal de la pentesting es identificar vulnerabilidades en el sistema que podrían ser explotadas por atacantes malintencionados.

Algunos aspectos clave de la pentesting incluyen:

- Identificación de vulnerabilidades: El pentester utiliza una variedad de técnicas y herramientas para identificar posibles vulnerabilidades en el

sistema objetivo. Esto puede incluir el escaneo de puertos, análisis de vulnerabilidades, revisión de código, entre otros métodos.
- Explotación de vulnerabilidades: Una vez que se identifican las vulnerabilidades, el pentester intenta explotarlas para demostrar el impacto potencial que podrían tener en el sistema. Esto puede incluir la ejecución de código malicioso, el acceso no autorizado a datos sensibles o el compromiso completo del sistema.
- Informes y recomendaciones: Después de completar las pruebas, el pentester elabora un informe detallado que describe las vulnerabilidades encontradas, el nivel de riesgo asociado y recomendaciones para mitigar o solucionar estos problemas de seguridad. Estos informes son utilizados por las organizaciones para mejorar su postura de seguridad y proteger sus activos contra posibles ataques.

El pentesting es una parte crucial de la estrategia de seguridad cibernética de una organización, ya que ayuda a identificar y remediar vulnerabilidades antes de que sean explotadas por atacantes reales. Sin embargo, es importante realizar las pruebas de manera ética y con el consentimiento del propietario del sistema objetivo para evitar consecuencias negativas no deseadas.

Lateral Movement

El "Lateral Movement" es una técnica utilizada por los atacantes cibernéticos para moverse lateralmente dentro de una red una vez que han comprometido un punto de entrada inicial. En lugar de atacar directamente el objetivo final, los atacantes buscan expandir su acceso y control dentro de la red, moviéndose de un sistema a otro para obtener información adicional, ampliar su presencia y aumentar las posibilidades de éxito en su objetivo final.

Algunos métodos comunes de "Lateral Movement" incluyen:

- Explotación de credenciales comprometidas: Los atacantes pueden utilizar credenciales robadas o comprometidas para autenticarse en otros sistemas dentro de la red.
- Explotación de vulnerabilidades: Los atacantes pueden aprovechar vulnerabilidades conocidas en sistemas o aplicaciones para comprometernos y moverse lateralmente dentro de la red.
- Uso de herramientas de administración remota: Los atacantes pueden utilizar herramientas de administración remota legítimas, como PowerShell remoto o RDP (Protocolo de escritorio remoto), para acceder y controlar sistemas dentro de la red.
- Intercepción de tráfico de red: Los atacantes pueden interceptar y redirigir el tráfico de red dentro de la red para obtener acceso a sistemas adicionales.

El "Lateral Movement" puede ser una fase crítica en un ataque cibernético, ya que permite a los atacantes mantener su presencia en la red y avanzar hacia sus objetivos finales mientras evitan la detección. Por lo tanto, es importante que las organizaciones implementen medidas de seguridad sólidas, como la segmentación de red, el monitoreo de tráfico y la autenticación multifactor, para mitigar el riesgo de "Lateral Movement" y proteger sus activos contra ataques cibernéticos.

Librerías

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Net;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.Linq;
using System.Threading.Tasks;
```

System: Esta es una biblioteca estándar de .NET que proporciona tipos y funciones básicas que son comunes en la mayoría de los programas C#. Contiene tipos fundamentales como String, Console, DateTime, Exception, entre otros.

System.Collections.Generic: Esta biblioteca proporciona tipos y clases genéricas que son esenciales para el almacenamiento y manipulación de colecciones de datos. Incluye clases como List<T>, Dictionary<TKey, TValue>, Queue<T>, entre otros.

System.Diagnostics: Esta biblioteca proporciona clases que permiten interactuar con el sistema operativo y controlar procesos, realizar seguimiento del tiempo y generar registros. En el código, se utiliza la clase Stopwatch para medir el tiempo transcurrido durante la ejecución del escaneo de red.

System.Net: Esta biblioteca proporciona clases y tipos para la comunicación en red, incluyendo acceso a Internet y conexiones TCP/IP. Se utiliza para realizar ping a dispositivos de red y escanear puertos abiertos en esos dispositivos.

System.Net.NetworkInformation: Esta biblioteca proporciona acceso a información y funcionalidades relacionadas con la configuración y el estado de las interfaces de red del sistema. Se utiliza para obtener la dirección IP y la máscara de subred del enrutador Wi-Fi.

System.Net.Sockets: Esta biblioteca proporciona clases que permiten la comunicación a nivel de sockets, como clientes y servidores TCP/IP. Se utiliza para escanear los puertos abiertos en los dispositivos de red mediante conexiones TCP.

System.Linq: Esta biblioteca proporciona funcionalidades para consultas y operaciones sobre colecciones de datos en lenguaje integrado de consultas (LINQ). Se utiliza en el código para realizar operaciones como filtrado y proyección sobre secuencias de datos.

Funciones

- Main: Esta es la función principal del programa. Comienza registrando el tiempo de inicio utilizando un Stopwatch. Luego, obtiene la dirección IP y la máscara de subred del enrutador Wi-Fi. Basándose en esta información, realiza un escaneo de red para detectar hosts activos y sus puertos abiertos(basado en el listado de puertos definidos más adelante). Una vez completado el escaneo, detiene el temporizador y calcula la duración total del escaneo en segundos.

- IsHostActive: Esta función realiza un ping ICMP a una dirección IP dada para determinar si el host está activo y responde al ping. Retorna true si el host responde correctamente, de lo contrario, retorna false.

```
static bool IsHostActive(string ipAddress)
{
    try
    {
        Ping ping = new Ping();
        PingReply reply = ping.Send(ipAddress, 100);

        return reply.Status == IPStatus.Success;
    }
    catch (Exception ex)
    {
        return false;
    }
}
```

- CheckOpenPorts: Esta función escanea una lista predefinida de puertos comunes en un host específico utilizando conexiones TCP. Registra los puertos abiertos encontrados en el host.

```
static void CheckOpenPorts(string ipAddress)
{
    List<int> openPorts = new List<int>();

    Parallel.ForEach(GetCommonPorts(), port =>
    {
        try
        {
            using (TcpClient tcpClient = new TcpClient())
            {
                tcpClient.Connect(ipAddress, port);
                openPorts.Add(port);
            }
        }
        catch (SocketException)
        {
            // Port is closed or unreachable
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error checking port {port} on host {ipAddress}: {ex.Message}");
        }
    });

    if (openPorts.Any())
    {
        Console.WriteLine($"Open ports on host {ipAddress}: {string.Join(", ", openPorts)}");
    }
}
```

- GetCommonPorts: Esta función devuelve una lista de puertos TCP comunes que se utilizarán en el escaneo de puertos. Definida manualmente por simplicidad, ya que al escanear los 1024 ports el programa se tardó más de una hora. Permite la adición de más puertos de manera rápida.

```
static IEnumerable<int> GetCommonPorts()
{
    return new List<int> { 21, 22, 23, 25, 53, 80, 110, 135, 139, 143, 443, 445, 993, 995 };
}
```

- GetRouterIpAndSubnet: Esta función obtiene la dirección IP y la máscara de subred del enrutador Wi-Fi al que está conectado el dispositivo. Utiliza la clase NetworkInterface para obtener esta información.

```
static void GetRouterIpAndSubnet(out string routerIpAddress, out string subnetMask)
{
    routerIpAddress = null;
    subnetMask = null;
    try
    {
        NetworkInterface wifiInterface = NetworkInterface.GetAllNetworkInterfaces()
            .FirstOrDefault(x => x.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 && x.OperationalStatus == OperationalStatus.Up);

        if (wifiInterface == null)
        {
            throw new InvalidOperationException("Wi-Fi interface not found or not operational");
        }

        IPInterfaceProperties wifiIpProperties = wifiInterface.GetIPProperties();
        GatewayIPAddressInformation gatewayAddress = wifiIpProperties.GatewayAddresses.FirstOrDefault();
        if (gatewayAddress == null)
        {
            throw new InvalidOperationException("Gateway address not found");
        }

        routerIpAddress = gatewayAddress.Address.ToString();
        subnetMask = wifiIpProperties.UnicastAddresses.FirstOrDefault(x => x.Address.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)?.IPv4Mask.ToString();
        if (subnetMask == null)
        {
            throw new InvalidOperationException("Subnet mask not found");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error retrieving router IP address or subnet mask: {ex.Message}");
    }
}
```

- GetBaseIpAddress: Esta función calcula la dirección base de la subred utilizando la dirección IP del enrutador y su máscara de subred. Es útil para determinar el rango de direcciones IP a escanear en la red.

```
static string GetBaseIpAddress(string routerIpAddress, string subnetMask)
{
    try
    {
        IPAddress routerIp = IPAddress.Parse(routerIpAddress);
        IPAddress mask = IPAddress.Parse(subnetMask);

        byte[] routerBytes = routerIp.GetAddressBytes();
        byte[] maskBytes = mask.GetAddressBytes();

        byte[] baseBytes = new byte[4];
        for (int i = 0; i < 4; i++)
        {
            baseBytes[i] = (byte)(routerBytes[i] & maskBytes[i]);
        }

        return new IPAddress(baseBytes).ToString();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error parsing router IP address or subnet mask: {ex.Message}");
        return null;
    }
}
```

- GetStartRange: Esta función calcula el primer valor en el rango de direcciones IP a escanear en la red. Utiliza la máscara de subred para determinar este valor.

```
static int GetStartRange(string subnetMask)
{
    try
    {
        IPAddress mask = IPAddress.Parse(subnetMask);
        byte[] maskBytes = mask.GetAddressBytes();

        // Calculate the network address
        byte[] networkAddressBytes = new byte[4];
        for (int i = 0; i < 4; i++)
        {
            networkAddressBytes[i] = (byte)(maskBytes[i] & maskBytes[i]);
        }

        return BitConverter.ToInt32(networkAddressBytes.Reverse().ToArray(), 0) + 1;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error parsing subnet mask: {ex.Message}");
        return 1;
    }
}
```

- GetEndRange: Esta función calcula el último valor en el rango de direcciones IP a escanear en la red. Utiliza la máscara de subred para determinar este valor.

```
static int GetEndRange(string subnetMask)
{
    try
    {
        IPAddress mask = IPAddress.Parse(subnetMask);
        byte[] maskBytes = mask.GetAddressBytes();

        int endRange = 0;
        for (int i = 0; i < 4; i++)
        {
            endRange += 255 - maskBytes[i];
        }

        return endRange;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error parsing subnet mask: {ex.Message}");
        return 255;
    }
}
```