Ataques a Servidores Web

Seguridad en los Sistemas de Información

Dra. Maricela Bravo

Servidores

- Un servidor es un programa que se ejecuta en computadoras normalmente más poderosas que las computadoras personales.
- Se ejecuta sobre sistemas operativos que soportan concurrencia, paralelismo y multiprogramación, por ejemplo Windows 200x server, o basados en Unix.

2

Tipos de Servidores

a. **Servidor de archivos**. Proporciona acceso a sistemas de archivos distribuidos. Los clientes pueden buscar en directorios, leer y escribir bloques de archivos, etc.

b. **Servidor de bases de datos**. Proporcionan acceso a uno o más DBMS. Las solicitudes de los clientes se realizan normalmente mediante lenguaje SQL.

c. **Servidor de aplicaciones**. Proporcionan acceso a procedimientos remotos, mediante la invocación de los clientes.

d. **Servidor de correo**. Ofrece servicio de envío y recepción de mensajes de correo, así como mensajería instantánea.

e. **Servidor Web**.

3

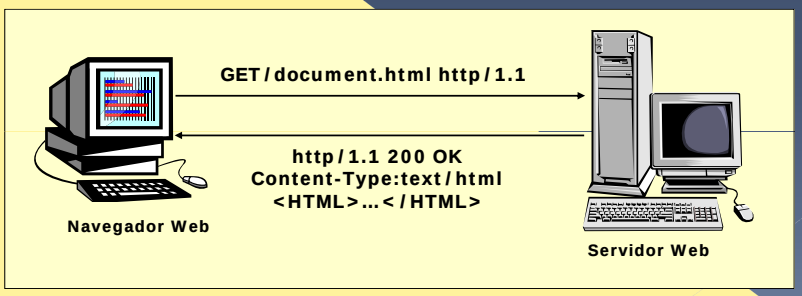
¿Qué es un Servidor Web?

- Un servidor Web o demonio HTTP es un programa que controla el flujo de datos entrantes y salientes de una computadora conectada a Intranet e Internet.
- Un servidor Web es un programa de aplicación que atiende las solicitudes HTTP realizadas por los navegadores.
- Escucha peticiones en el número de puerto 80, normalmente.
- Los programas de aplicación más difundidos para montar un servidor Web son:
  - Apache Tomcat
  - Internet Information Server

4

Protocolo HTTP

- Es un protocolo de petición/respuesta sin estado cuya operación básica es la siguiente :



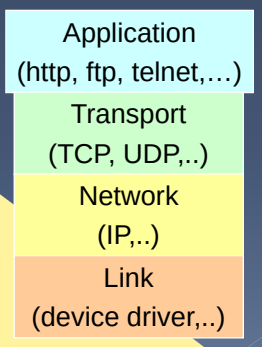
5

Sockets

Pila de protocolos TCP/IP

- **Capa de Aplicación**
  - Aplicaciones estándar
    - HTTP
    - FTP
    - Telnet
  - Aplicaciones de usuario
- **Capa de Transporte**
  - TCP
  - UDP
  - Interfaces de programación:
    - Sockets
- **Capa de Red**
  - IP
- **Capa de Enlace**
  - Drivers de dispositivos

- TCP/IP Stack

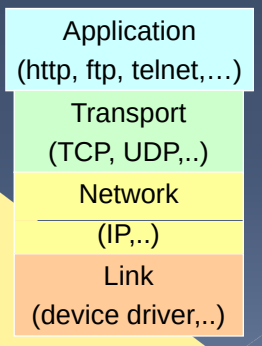


7

Pila de protocolos TCP/IP

- TCP (Transport Control Protocol)
- Es un protocolo orientado a conexión que proporciona un flujo de datos confiable entre dos computadoras.
- Ejemplo de aplicaciones:
  - HTTP
  - FTP
  - Telnet

- TCP/IP Stack

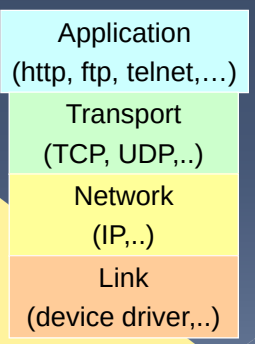


8

Pila de protocolos TCP/IP

- UDP (User Datagram Protocol)
- Es un protocolo que envía paquetes de datos independientes, llamados datagramas, de una computadora a otra, sin garantizar su llegada.
- Ejemplo de aplicaciones:
  - Clock server
  - Ping

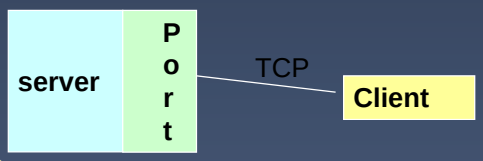
- TCP/IP Stack

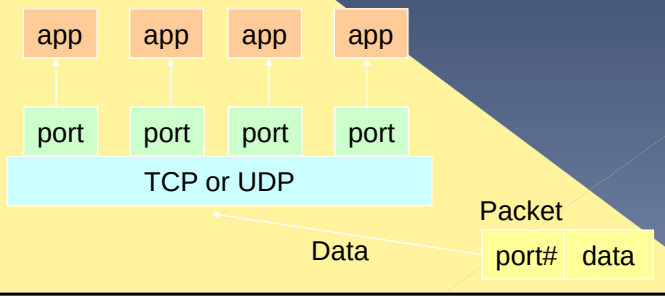


9

¿Qué son los puertos?

- TCP y UDP utilizan puertos para enviar datos entrantes a un proceso particular que se esté ejecutando en la computadora.





¿Qué son los puertos?

- Los puertos son representados por valores enteros positivos de 16 bits.
- Algunos puertos están reservados para soportar servicios preestablecidos:
  - FTP 21/TCP
  - Telnet 23/TCP
  - SMTP 25/TCP
  - HTTP 80/TCP
- Los procesos o servicios de usuarios generalmente usan números de puertos >= 1024.

11

Sockets

- Los sockets proporcionan una interfaz para la programación de redes en la capa de transporte.
- Las comunicaciones de redes utilizando Sockets es muy similar al manejo de I/O en archivos.
  - De hecho, el manejo de sockets es tratado como el manejo de archivos.
  - Los streams utilizados en operaciones de I/O de archivos también son aplicables a I/O basado en sockets.
- La comunicación basada en Sockets es independiente del lenguaje de programación.
  - Esto es, que un programa de socket escrito en Java también se puede comunicar con un programa escrito en Java o con un programa de socket no escrito en Java.

12

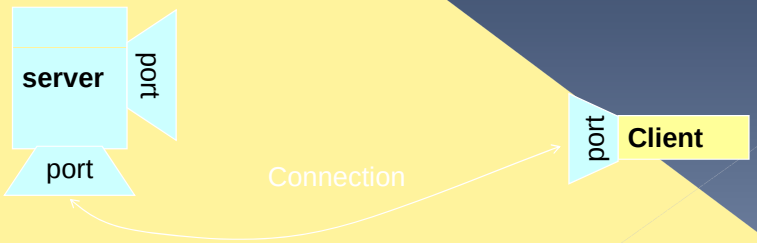
Comunicación entre Sockets

- Un servidor (programa) corre en una computadora específica y tiene un socket que se asocia con un puerto específico. El servidor se mantiene en espera escuchando al socket para cuando un cliente realiza una petición de conexión.

13

Comunicación entre Sockets

- Si todo sale bien, el servidor acepta la conexión. Después de la aceptación, el servidor obtiene un nuevo socket asociado a un puerto diferente. Necesita un nuevo socket (consecuentemente un número de puerto diferente), de tal forma que puede continuar escuchando al socket original para solicitudes de conexión mientras que atiende al cliente conectado.



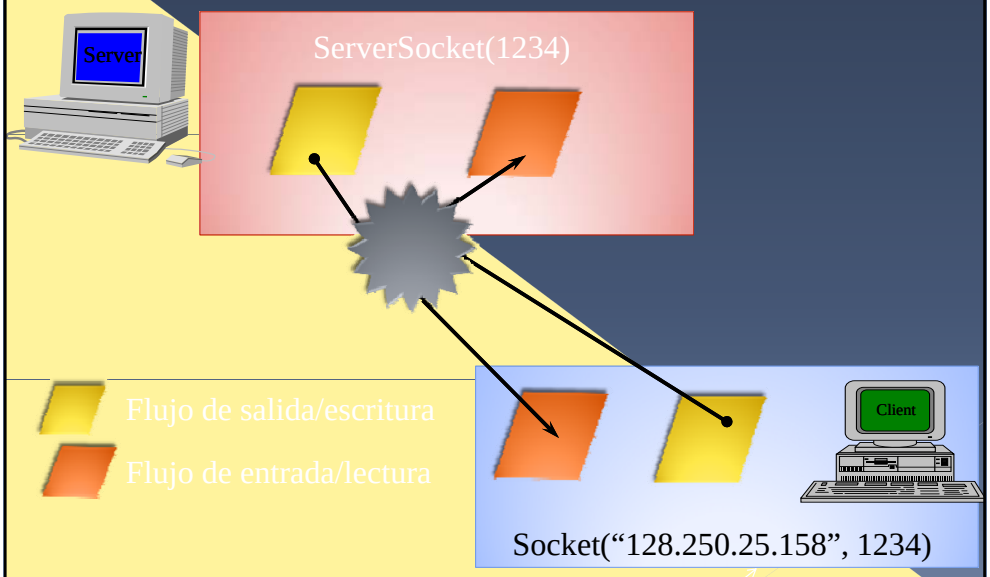
14

Clases de Java para crear Sockets

- Un socket es un endpoint de un enlace de comunicación bi-direccional entre dos programas ejecutándose en la red.
- Un socket se asocia a un número de puerto de tal forma que la capa de TCP puede identificar la aplicación a la cual están destinados los datos.
- El paquete de Java .net proporciona dos clases:
  - Socket – para implementar un cliente
  - ServerSocket – para implementar un servidor.

15

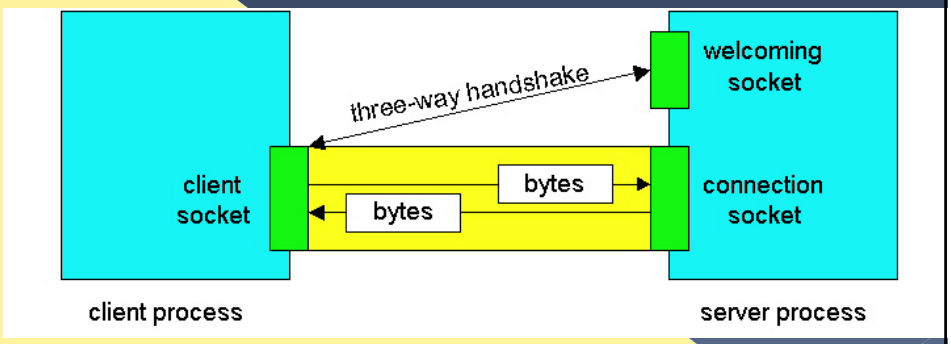
Java Sockets



Puede ser un dominio como "mandroo.cs.mu.oz.au"

16

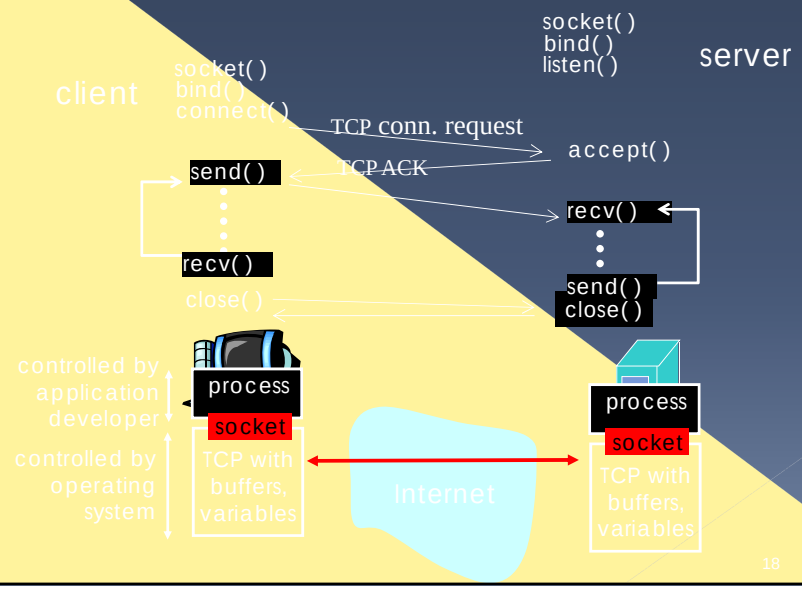
Sockets



17

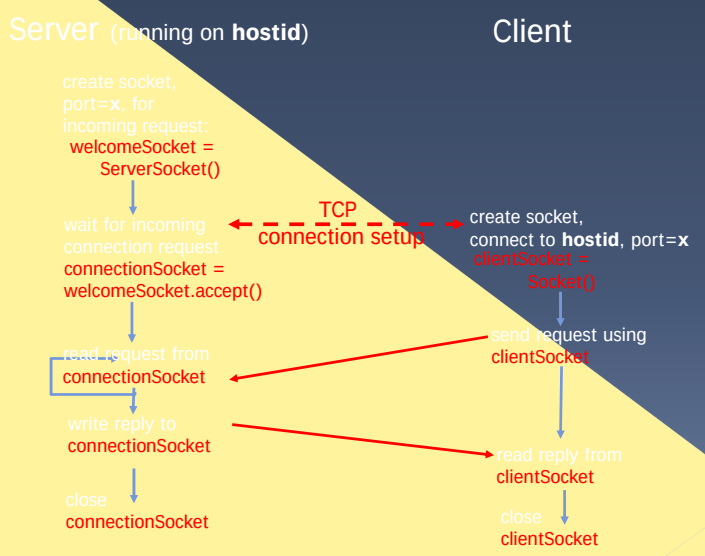
Programación de Sockets usando TCP

TCP service: transferencia confiable de flujos de bytes



18

Client/server socket interaction: TCP



19

Ejemplo de un Server

1. Crear el Server Socket:

```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```

2. Espera solicitudes de clientes:

```
Socket client = server.accept();
```

3. Crea flujos de I/O para comunicarse con el cliente

```
is = new DataInputStream( client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```

4. Realiza comunicación con un cliente

```
Receive from client: String line = is.readLine();
Send to client: os.writeBytes("Hello\n");
```

5. Cierra el socket: `client.close();`

20

Ejemplo de un Cliente

1. Crear un objeto de Socket:
   `client = new Socket( server, port_id );`
2. Crea flujos de I/O para comunicarse con el servidor.
   `is = new DataInputStream(client.getInputStream() );`
   `os = new DataOutputStream( client.getOutputStream() );`
3. Realiza I/O o comunicación con el server:
   - Receive data from the server:
     `String line = is.readLine();`
   - Send data to the server:
     `os.writeBytes("Hello\n");`
4. Cierra el socket cuando termina:
   `client.close();`

21

Un Server Simple

```
import java.net.*;
import java.io.*;

    public class SimpleServer
    {
      public static void main(String args[]) throws IOException
      {
        // Registrar el servicio en el puerto 1234
        ServerSocket s = new ServerSocket(1245);
        //Espera y acepta conexiones
        Socket s1 = s.accept();
        //Obtiene un flujo de comunicación asociado con el socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (s1out);
        //Envia un mensaje
        dos.writeUTF("Hola que tal");
        //Cierra la conexión, pero no el socket del servidor
        dos.close();
        s1out.close();
        s1.close();
      }

}
```

22

Un Cliente Simple

```
import java.net.*;
import java.io.*;

public class SimpleClient
{
  public static void main(String args[]) throws IOException
  {
    //Abrir una conexión al server en el puerto 1234
    Socket s1 = new Socket("localhost",1245);
    //Obtener un manejador de flujo de entrada del socket y leer la entrada
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new DataInputStream(s1In);
    String st = new String (dis.readUTF());
    System.out.println(st);
    //Cerrar la conexion
    dis.close();
    s1In.close();
    s1.close();
  }
}
```

23

Ejecución

- Ejecutar Server en el localhost
  - java SimpleServer
- Ejecutar el Client en cualquier máquina:
  - java SimpleClient
    Hola que tal
- Si se ejecuta el cliente cuando el server no está escuchando:
  - java SimpleClient

```
Exception in thread "main" java.net.ConnectException: Connection refused
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
    at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
    at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
    at java.net.Socket.<init>(Socket.java:273)
    at java.net.Socket.<init>(Socket.java:100)
    at SimpleClient.main(SimpleClient.java:6)
```

24

Servidor Web

- Maneja solamente una petición HTTP
- Acepta y parsea la petición HTTP
- Obtiene el archivo requerido del sistema de archivos del servidor
- Crea un mensaje de respuesta HTTP, el cual consiste del archivo precedido por líneas de cabecera.
- Envía la respuesta directamente al cliente.

25

```
import java.io.*;
import java.net.*;
import java.util.*;

public class WebServer
{

  public static void main(String[] args)
  {
    String requestMessageLine;
    String fileName;
    try
     {
       ServerSocket listenSocket = new ServerSocket(8000);
       Socket connectionSocket = listenSocket.accept();

       BufferedReader inFromClient = new BufferedReader(
              new InputStreamReader(connectionSocket.getInputStream()));

       DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());

       requestMessageLine = inFromClient.readLine();
       StringTokenizer tokenizedLine = new StringTokenizer(requestMessageLine);

              if (tokenizedLine.nextToken().equals("get"))
              {
               fileName = tokenizedLine.nextToken();
                          if (fileName.startsWith("/") == true )
                            fileName = fileName.substring(1);
```

26

```
File file = new File(fileName);

int numOfBytes = (int) file.length();
FileInputStream inFile = new FileInputStream (fileName);
byte[] fileInBytes = new byte[numOfBytes];
inFile.read(fileInBytes);
outToClient.writeBytes("HTTP/1.1 200 Document Follows\r\n");

if (fileName.endsWith(".jpg"))
        outToClient.writeBytes("Content-Type: image/jpeg\r\n");

if (fileName.endsWith(".gif"))
        outToClient.writeBytes("Content-Type: image/gif\r\n");
```

27

```
outToClient.writeBytes("Content-Length: " + numOfBytes + "\r\n");
outToClient.writeBytes("\r\n");
outToClient.write(fileInBytes, 0, numOfBytes);
connectionSocket.close();
}
else System.out.println("Bad Request Message");
}
catch (IOException e)
{
        e.printStackTrace();
}
}

}
```

28

```
/* ChatServer.java */
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
class ChatServer {
private static int port = 1001; /* port the server listens on */
public static void main (String[] args) throws IOException {
ServerSocket server = null;
try {
server = new ServerSocket(port); /* start listening on the port */
 } catch (IOException e) {
System.err.println("Could not listen on port: " + port);
System.err.println(e); System.exit(1); }
Socket client = null;
```

```
try {
 client = server.accept();
 } catch (IOException e) {
 System.err.println("Accept failed.");
 System.err.println(e); System.exit(1); }   /* obtain an input stream to the
 client */ BufferedReader in = new BufferedReader(new InputStreamReader(
 client.getInputStream()));
 String msg;
 /* loop reading lines from the client and display them */
 while ((msg = in.readLine()) != null) {
 System.out.println("Client says: " + msg); } } }
```

```
/* ChatClient.java */
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;   class ChatClient { private static int
port = 1001; /* port to connect to */ private static String host = "localhost"; /*
host to connect to */   public static void main (String[] args) throws
IOException {   Socket server; PrintWriter out = null;   try { /* try to open a socket
to the server at the given host:port */ server = new Socket(host, port); /* obtain
an output stream to the server */ out = new
PrintWriter(server.getOutputStream(), true); } catch (UnknownHostException e)
{ System.err.println(e); System.exit(1); }   BufferedReader stdIn = new
BufferedReader( new InputStreamReader(System.in)); String msg;   /* loop
reading lines from stdin and output what was read * to the server */ while
((msg = stdIn.readLine()) != null) { out.println(msg); } } }
```

Socket Exceptions

```
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) {
    System.out.println("Unknown host: " + host);
    uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe);
    ioe.printStackTrace();
}
```

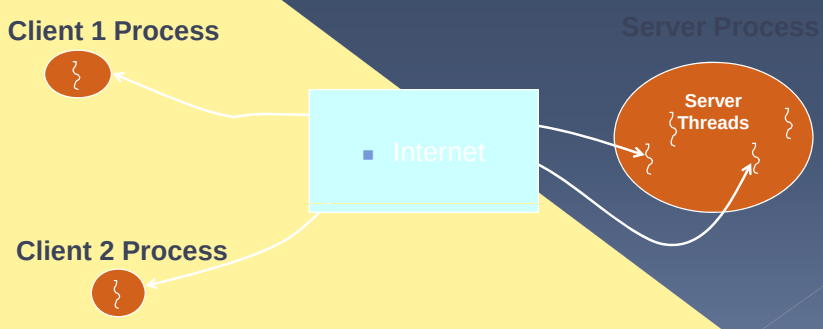
32

Servidor en un Ciclo: Siempre en Ejecución

```
import java.net.*;
import java.io.*;
public class SimpleServerLoop
{
  public static void main(String args[]) throws IOException
  {
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
          Socket s1=s.accept();
         OutputStream s1out = s1.getOutputStream();
          DataOutputStream dos = new DataOutputStream (s1out);
          dos.writeUTF("Hola");
          dos.close();
          s1out.close();
          s1.close();
    }
  }
}
```

33

Multithreaded Server: Para servir a múltiples clientes concurrentemente



34

Servidor de Eco con Hilos

```
import java.io.*;
import java.net.*;
import java.util.*;
public class servidorEcoconHilos
{
        public static void main(String[] args) {
          try {
            int i = 1;
             ServerSocket s = new ServerSocket(8189);
            while(true)
             {
                Socket entrante = s.accept();
                System.out.println("generando hilo " + i);
                Runnable r = new ManejadorHilos(entrante, i);
                Thread t = new Thread(r);
                t.start();
                i++;
             }
           }
```

35

```
catch(IOException e) {
                e.printStackTrace();
            }
        }
}
//Clase para el manejo de hilos
Class ManejadorHilos implements Runnable
{
  public ManejadorHilos(Socket i, int c)
  {
    entrante = i;
    contador = c;
  }

  public void run()
  {
    try {
        try {
          InputStream secuenciaEntrada = entrante.getInputStream();
          OutputStream secuenciaSalida = entrante.getOutputStream();
           Scanner in = new Scanner(secuenciaEntrada);
           PrintWriter out = new PrintWriter(secuenciaSalida, true);
           out.println("Escriba ADIOS para salir");
```

36

```
//Reproducir la entrada del cliente
boolean terminado = false;
while(!terminado && in.hasNextLine()) {
   String linea = in.readLine();
   out.println("Eco:  " + linea );
   if (linea.trim().equals("ADIOS"))
            terminado = true;
}
}
finally
{
   entrante.close();
}
Catch(IOException e)
{
    e.printStackTrace();
}
}//End del run

private Socket entrante;
private int contador;
}
```

37

Introducción a los ataques DoS

- Los ataques de **Denegación de Servicio (DoS)** tienen la finalidad de provocar que un servicio o recurso sea innacesible para los usuarios legítimos.
- Este tipo de ataques pueden provocar:
  - Parada de todos los servicios de una máquina
  - La máquina sólo puede dar determinados servicios
  - La máquina no puede dar servicio a determinados usuarios

Introducción a los ataques DoS

Modos de ataque

Los ataques DoS se pueden llevar a cabo de diferentes formas y cubren infinidad de servicios. Existen tres tipos básicos de ataque:

- Consumo de recursos limitados.
- Destrucción o alteración de datos.
- Destrucción o alteración física de componentes de la red.

Introducción a los ataques DoS

Atacantes

Algunos de los grupos que pueden llevar a cabo este tipo de ataques son:

- Script Kiddies
- Competencia
- Militares
- Empleados incompetentes

Introducción a los ataques DoS

Ejemplos de ataques DoS

- Consumo de ancho de banda:
  - Smurf Attack
  - ICMP Ping Flood
  - Fraggle Attack
- Ataques a la conectividad
  - SYN Flood Attacks

Introducción a los ataques DoS

Consumo de Ancho de Banda

- **Smurf Attack** : Este ataque se basa en mandar un gran número de peticiones echo (ICMP) a direcciones de Broadcast usando una IP de origen falsa. Esto provoca que la IP de origen sea inundada con multitud de respuestas.
- **ICMP Ping Flood:** En este ataque se inunda a la víctima con paquetes ICMP Echo Request.
- **Fraggle Attack:** Es similar al ataque Smurf pero en este caso se envía tráfico UDP en lugar de ICMP.

Introducción a los ataques DoS

Ataques a la conectividad

- **SYN Flood Attack:** Consiste en enviar muchos paquetes TCP/SYN con la dirección de origen falseada. Esto provoca que el servidor espere las respuestas que nunca llegan, provocando un consumo elevado de recursos que afectan al rendimiento del servidor.

Introducción a los ataques DDoS

- Un ataque de **Denegación de Servicio Distribuido (DdoS)** es un tipo especial de ataque DoS en el que se utilizan varios equipos para realizar un ataque coordinado contra una máquina.
- En este tipo de ataque se suelen utilizar máquinas denominadas Zombies que el atacante consigue controlar gracias a algún tipo de malware.
- Al conjunto de máquinas Zombies que controla un atacante se las suele denominar BotNets.

Introducción a los ataques DDoS

Existen multitud de herramientas de DdoS conocidas, algunas de las más importantes son:

- **Trinoo**: Primera herramienta conocida de este tipo.
- **TFN y TFN2K:** ICMP Flood, SYN Flood, UDP Flood.
- **Stacheldraht:** ICMP Flood, SYN Flood, UDP Flood y Smurf.
- **Shaft:** SYN flooding, UDP flooding, ICMP flooding, and Smurf

Introducción a los ataques DDoS

Ejemplos de ataques DdoS reales:

- En **Febrero de 2000** ualgunas compañías incluyendo Yahoo, Amazon y Ebay sufrieron una serie de ataques DdoS que les dejaron sin servicio.
- En **Julio de 2001** una serie de ataques DdoS tuvieron efecto sobre la web de la Casa Blanca.
- En **Octubre de 2002** se realizó un ataque DdoS contra los servidores DNS raíz quelograron afectar a 9 de 13 de los servidores DNS que controlan Internet.
- En **Mayo de 2007** Estonia sufrió un ataque DdoS masivo que afectó a multitud de servidores incluyendo la presidencia, el parlamento, varios ministerios, bancos, telecomunicaciones, etc.

¿Preguntas?, ¿Comentarios?

¡GRACIAS!