Arquitectura e Integración de Aplicaciones Empresariales

Octava Sesión
Seguridad para Aplicaciones Empresariales

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Dra. Maricela Bravo
Cubículo H-287-B
mari_clau_18@hotmail.com

Principales obstáculos de la seguridad informática

Falta de conciencia de los usuarios finales…

Falta de apoyo por parte de la gerencia…

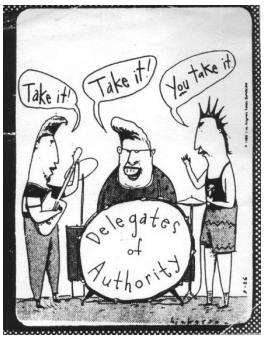
Falta de presupuesto…

Principales obstáculos de la seguridad informática

Falta de entrenamiento…

Responsabilidades no claras…

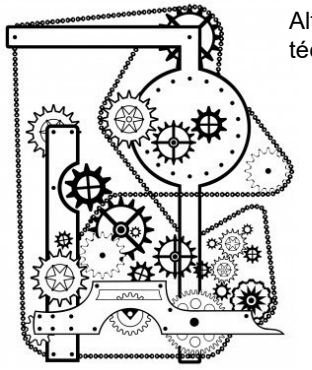
Principales obstáculos de la seguridad informática



Falta de una autoridad centralizada…

Falta de herrameintas de seguridad…

Principales obstáculos de la seguridad informática



Alta complejidad técnica…

Falta de estándares de seguridad…

Information security standards

Principales obstáculos de la seguridad informática

Falta de personal de seguridad competente…

Aspectos legales de la seguridad informática…

Legislación de la Seguridad Informática en México

- El Derecho surge como un medio efectivo para regular la conducta del hombre en sociedad.

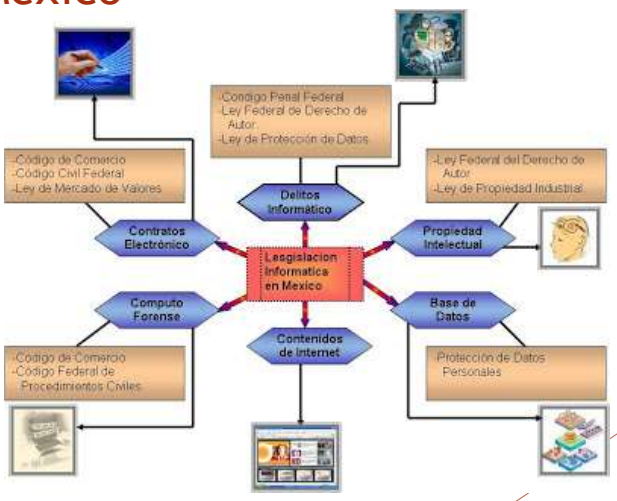
Pero la sociedad no es la misma en cada uno de los lugares del planeta ni es la misma en cada momento de la historia.

Legislación de la Seguridad Informática en México

El Derecho regula la conducta y los fenómenos sociales a través de leyes.

- El proceso de creación de las leyes es largo y lento, sobre todo en el Sistema Jurídico Latino.

Legislación Informática en México



¿Por qué es importante la seguridad en las aplicaciones empresariales?

- La información es un recurso esencial para todas las organizaciones
- La información puede ser la clave para el crecimiento de las organizaciones
- Compartir información es una práctica común hoy y continua incrementándose
- La información es uno de los principales activos de la organización
- La disponibilidad, integridad y confidencialidad de la información
  - puede ser crítica para el éxito de la organización
  - provee nuevas oportunidades de negocio
  - asegura lealtad de los clientes

Ataques a Servidores

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.

12

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**.

13

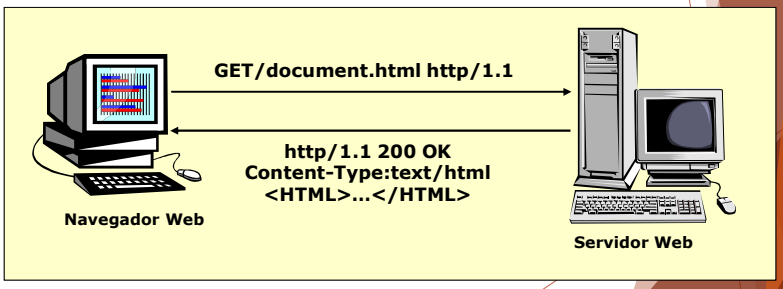
¿Qué es un Servidor Web?

- 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.

14

Protocolo HTTP

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



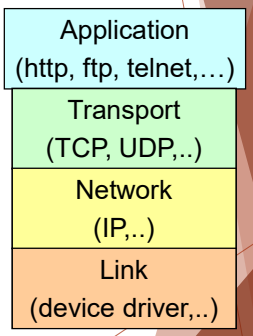
15

Repaso de 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

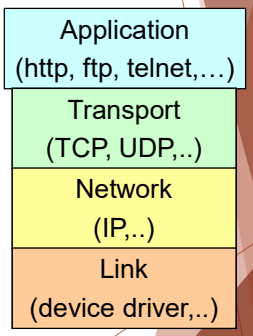


17

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

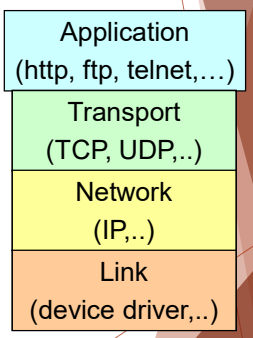


18

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

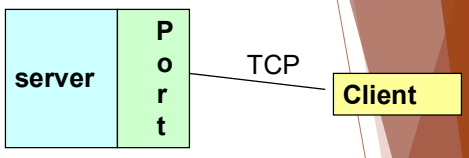
- TCP/IP Stack

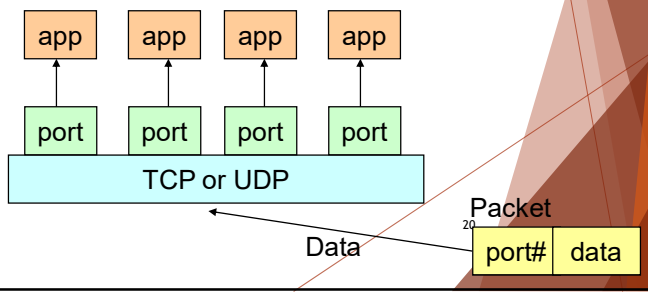


19

¿Qué son los puertos?

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





20

¿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.

21

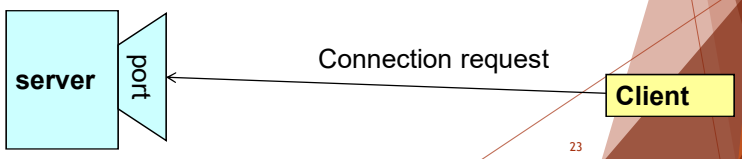
Sockets

- 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.

22

Comunicación entre Sockets

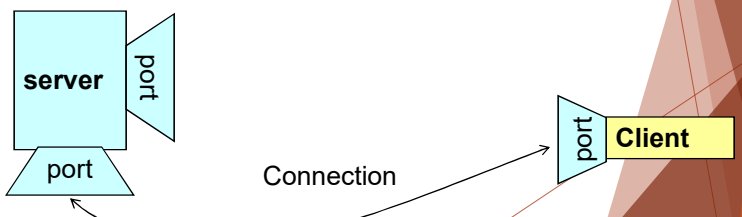
- Un servidor (programa) corre en una computadora específica y tiene un socket que se asocia con un puerto específico. El servidor se mantiene en espera escuchando al socket para cuando un cliente realiza una petición de conexión.



23

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.



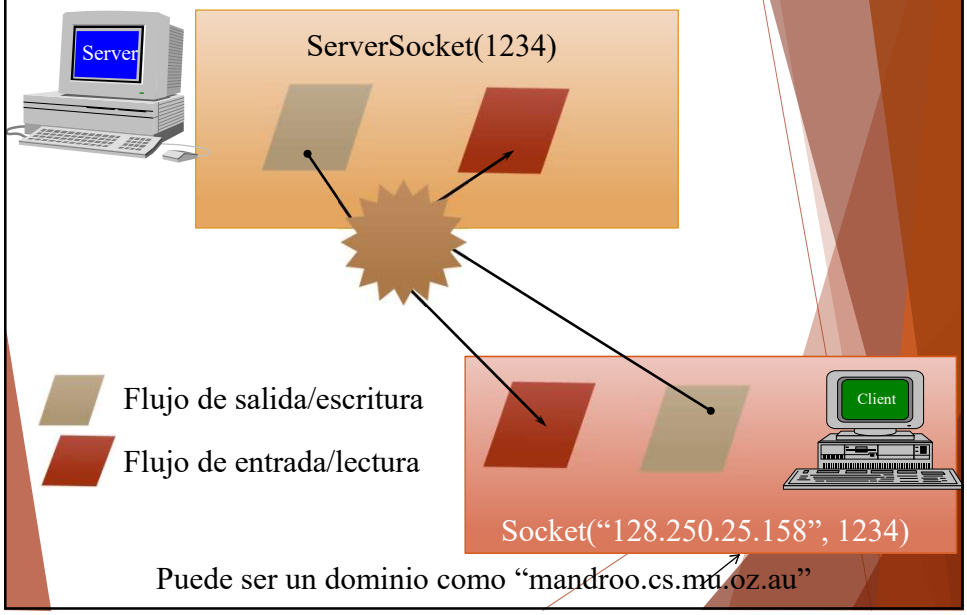
24

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.

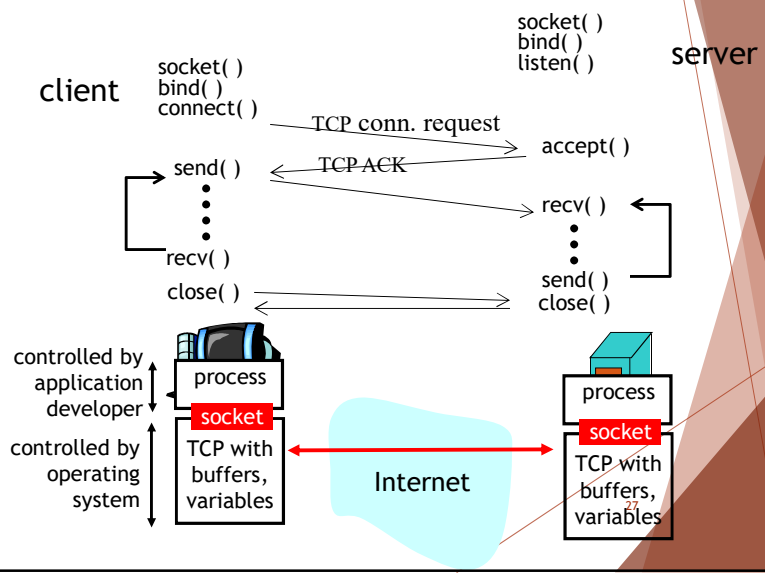
25

Java Sockets



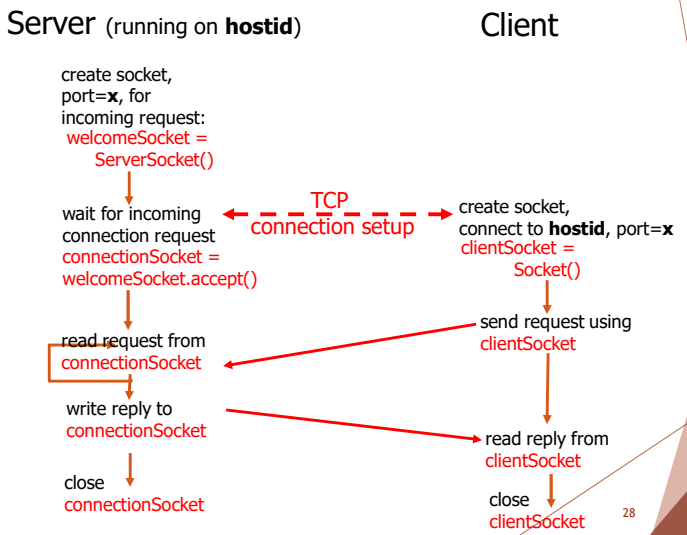
Programación de Sockets usando TCP

TCP service: transferencia confiable de flujos de bytes



27

Client/server socket interaction: TCP



28

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();`

29

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();
```

30

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();
       }

}
```

31

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();
  }
}
```

32

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)
```

33

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.

34

```
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);
```

35

```
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");
```

36

```
 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();
}
}

}
```

37

```
/* 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();
}
```

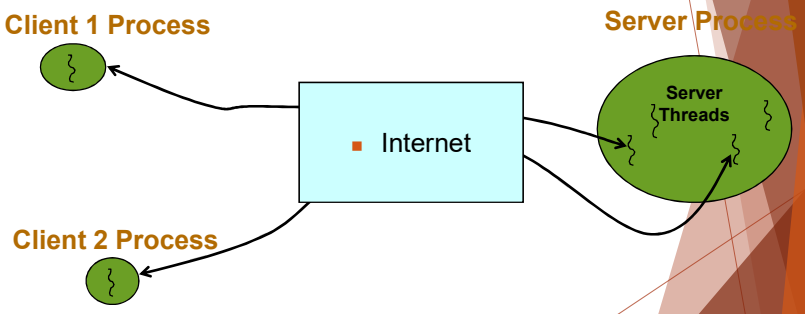
41

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();
    }
  }
}
```

42

Multithreaded Server: Para servir a múltiples clientes concurrentemente



43

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++;
             }
           }
```

44

```
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");
```

45

```
//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;
}
```

46

Tipos de Ataques a Servidores Web

Tipos de Ataques

- Ataque DoS. En un ataque de denegación de servicio (DoS), un atacante intenta evitar la legitimidad de que los usuarios accedan a información o al servicios.
- Ping Flood. Ping flood se basa en enviar a la víctima una cantidad abrumadora de paquetes ping.
- Escaneo de puertos. El escaneo de puertos es una de las técnicas de reconocimiento más populares que utilizan los atacantes para descubrir los servicios expuestos a posibles ataques.
- Ataque FTP Bounce. El atacante puede conectarse a los servidores FTP y tener la intención de enviar archivos a otros usuarios / máquinas que usan el comando PORT.
- TCP Session Hijacking. Es el caso cuando el "Hacker" toma la sesión TCP existente, ya establecido entre las dos partes. En la mayoría de las sesión TCP la autenticación ocurre al comienzo de la sesión.

Ataque DoS

- El tipo más común y obvio de ataque DoS ocurre cuando un atacante "inunda" una red con información. Cuando escribimos una URL de un sitio web en particular en nuestro navegador, estamos enviando una solicitud al servidor web del sitio para poder ver la página en concreto.
- El servidor solo puede procesar una cierta cantidad de solicitudes de una vez, por lo que si un atacante sobrecarga el servidor con solicitudes, no puede procesarse dicha solicitud. Esto es una "denegación de servicio" ya que no se puede acceder al sitio.

Implementación del ataque DoS

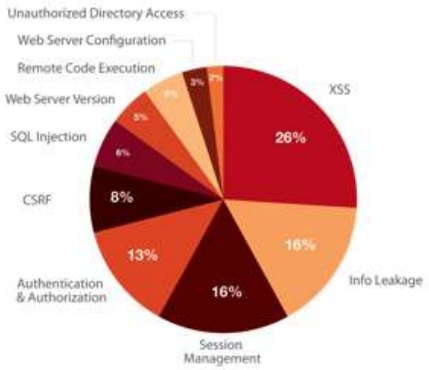
Seguridad del servidor Web Apache

Introducción

- El servidor Web es una parte crucial de las aplicaciones Web.
- El servidor Web Apache es uno de los más atacados.
- La mayoría de los ataques a aplicaciones Web ocurren por:
  - XSS
  - Fuga de información
  - Mal manejo de sesiones
  - Inyección de código PHP.

Introducción

- De acuerdo con Cenzic, 99% de las aplicaciones Web probadas presentan vulnerabilidades (2013).



XSS - Cross-site scripting

- **XSS**, del inglés ***Cross-site scripting*** es un tipo de agujero de seguridad típico de las aplicaciones Web, que permite a una tercera parte inyectar en páginas web vistas por el usuario código JavaScript o en otro lenguaje script similar (ej: VBScript).

XSS – Cross-site scripting

- Causada por no validar correctamente los datos de entrada que son usados en la aplicación, o no sanear la salida adecuadamente para su presentación como página web.
- Esta vulnerabilidad puede estar presente de las siguientes formas:
  - **Directa** (también llamada Persistente): este tipo de XSS comúnmente filtrado, y consiste en embeber código HTML peligroso en sitios que lo permitan; incluyendo así etiquetas como <script> o <iframe>.
  - **Indirecta** (también llamada Reflejada): este tipo de XSS consiste en modificar valores que la aplicación web utiliza para pasar variables entre dos páginas, sin usar sesiones y sucede cuando hay un mensaje o una ruta en la URL del navegador, en una *cookie*, o cualquier otra cabecera HTTP .

Métodos para pasar variables entre dos páginas y sus vulnerabilidades

Métodos para el rastreo de variables en aplicaciones Web

- ¿Por qué realizar el rastreo de datos o variables?
- Formas de rastreo
  - Cookies
  - Reescritura de URLs
  - Campos ocultos
- Sesiones (Session Tracking)

¿Por qué realizar el rastreo de sesiones?

- Cuando los clientes de una tienda *on-line* añaden artículos a su carrito de compras, ¿cómo sabe el servidor lo que hay ya en sus carritos de la compra?
- Cuando los clientes deciden confirmar el pedido, ¿cómo sabe el pedido cuál de los carritos de compras previamente creados es el suyo?
- En un Sistema de Información Empresarial, es importante saber qué usuario está realizando operaciones para adjudicarle un "*rol*" y permitirle ciertas operaciones y otras no.

Formas de realizar el rastreo de sesiones

- HTTP es un protocolo "*sin estado*"
  - Cada vez que un cliente pide una página Web, abre una conexión separada con el servidor Web y el servidor no mantiene automáticamente *información contextual* acerca del cliente
- Servlets
  - Permiten obtener y mantener una determinada información acerca de un cliente
  - Información accesible a diferentes servlets o entre diferentes ejecuciones de un mismo servlet
- Tres soluciones típicas
  - Cookies
  - Reescritura de URLs
  - Campos ocultos de formularios

Cookies

- Objetos de la clase Cookie
  - Permite guardar información relativa a un usuario a lo largo de sus distintos accesos
- Se almacenan en los equipos de los clientes
  - El cliente debe soportar cookies
  - Pueden ser desactivadas por el cliente
  - El navegador es el encargado de almacenarlas
- Se transmiten en las cabeceras cuando se realiza la comunicación HTTP.

Uso de las Cookies

- Identificación del usuario durante una sesión
  - Los servlets ofrecen una API de alto nivel para realizar esta tarea.
- Para evitar tener que porporcionar el usuario y password cada vez
- Personalización del sitio
- Publicidad enfocada

Problemas con las Cookies

- El problema es la privacidad y la seguridad
  - Los servidores pueden recordar todas tus acciones previas.
  - Si proporcionas información personal, los servidores pueden vincular esa información con tus acciones previas.
  - Los servidores pueden compartir la información recolectada en las cookies con terceros.
  - Algunos sitios almacenan información sensible como los números de tarjeta de crédito en una cookie.
- Notas para los autores de servlets
  - Si las cookies no son críticas para tu tarea, evita los servlets que fallan totalmente cuando las cookies están deshabilitadas.
  - No coloques información sensible en las cookies.

Reescritura de URLs

- Idea
  - El cliente añade ciertos datos extra que identifican la sesión al final de cada URL
    - `http://host/path/servlet/name?jsessionid=1234`
  - El servidor asocia ese identificador con datos que ha guardado acerca de la sesión
- Ventajas
  - Funciona incluso si las Cookies no son soportadas o están desactivadas
- Desventajas
  - Se deben codificar todas las URLs referentes al sitio propio
  - Todas las páginas deben generarse dinámicamente
  - Funciona mal para links desde otros sitios
  - El servletrunner no soporta reescritura de URLs

Campos ocultos de formularios

- Idea
  - Incluir campos ocultos con los datos a mantener

```
<INPUT type="hidden" name="session" value="1234">
```

- Ventajas
  - Funciona incluso si las Cookies no son soportadas o están desactivadas
- Desventajas
  - Cantidad de procesamiento tedioso
  - Todas las páginas deben ser el resultado de envios de formularios

Sesiones en Java (Session Tracking)

- Los objetos de la sesión se guardan en el servidor
- Se pueden guardar objetos arbitrarios dentro de una sesión
- Las sesiones se asocian automáticamente al cliente vía Cookies o Reescritura de URLs
  - Como una caja negra para el cliente, el sistema se encarga de utilizar el método apropiado para mantener la sesión, bien mediante cookies o mediante reescritura de URLs.