FACULTAD DE INGENIERÍA

**Universidad Nacional de Jujuy**

Unidad 6

# Seguridad en Aplicaciones Web

Apunte de Cátedra · Seguridad y Auditoría Informática

# Contenidos de la Unidad 6

**01** **Introducción y Arquitectura Cliente-Servidor**
Importancia, componentes, flujo de petición y riesgos

**02** **Principales Ataques Web**
SQL Injection, XSS, CSRF, Session Hijacking, fuerza bruta, archivos, DoS/DDoS

**03** **OWASP Top 10**
Qué es OWASP y las categorías principales

**04** **Autenticación y Gestión de Sesiones**
Protección de sesiones, cookies y buenas prácticas

**05** **Protección de Aplicaciones**
Firewall tradicional, WAF, HTTPS y certificados

**06** **Buenas Prácticas**
Principios y medidas fundamentales de seguridad web

# 6.1 Introducción a la Seguridad en Aplicaciones Web

Las aplicaciones web se ejecutan sobre servidores conectados a Internet y pueden ser utilizadas por miles o millones de usuarios desde cualquier lugar. Esto aumenta la superficie de ataque.

Finalidad: proteger la aplicación y la información frente a accesos no autorizados, modificaciones indebidas, robo de información o interrupciones del servicio.

## Consecuencias de una falla de seguridad en una aplicación web

| | | |
|---|---|---|
| Pérdida de información confidencial | Robo de datos personales | Fraude económico |
| Interrupción del servicio | Pérdida de confianza | Sanciones legales |

**La seguridad debe considerarse desde las primeras etapas del desarrollo de una aplicación y mantenerse durante todo su ciclo de vida**

# 6.1.1–6.1.2 Importancia y objetivos

## ¿Por qué requieren seguridad específica?

- Disponibles 24 horas
- Accesibles desde cualquier lugar
- Solicitudes de usuarios desconocidos
- Procesan información sensible
- Se integran con otros sistemas y BD

**Objetivos: los mismos del modelo CIA**
**Confidencialidad · Integridad · Disponibilidad**

# 6.2 Arquitectura Cliente-Servidor

Modelo en el cual un programa cliente solicita servicios a un programa servidor. En aplicaciones web el cliente suele ser el navegador; el servidor ejecuta la aplicación y administra la base de datos.

### Navegador

Interfaz del usuario. Envía solicitudes y muestra respuestas.

### Servidor Web

Recibe peticiones, entrega páginas, administrar recursos estáticos, comunicarse con el servidor de aplicaciones Ej.: Apache, Nginx, IIS.

### Servidor de Aplicaciones

Lógica de negocio, validación, consultas a BD, generar respuesta hacia el navegador. Ej.: Tomcat, WebLogic, Gunicorn, PM2.

### Base de Datos

Almacena usuarios, datos, operaciones. Objetivo crítico de protección.

# 6.2.3 Flujo de una Petición Web

1. Usuario realiza una acción
2. Navegador envía solicitud (HTTP/HTTPS)
3. Servidor web recibe y deriva al servidor de aplicaciones
4. Servidor de aplicaciones procesa la información (valida, autenticación, comprueba permisos, consulta o modifica información almacenada)
5. Acceso a la base de datos
6. Generación de la respuesta (pagina web, lista de productos, etc.)
7. Navegador muestra el resultado

# 6.2.4 Riesgos asociados a cada componente

| Componente | Principales riesgos |
|---|---|
| Navegador | Phishing, XSS, robo de sesiones |
| Servidor Web | Configuración incorrecta, software desactualizado , exposición innecesaria de servicios |
| Servidor de Aplicaciones | Validación incorrecta de datos, control de acceso insuficiente, manejo incorrecto de sesiones, errores de programación |
| Base de Datos | Robo, acceso no autorizado, modificación o eliminación de información |

La seguridad no depende solo del servidor o de la BD: todos los componentes deben protegerse de forma conjunta.

# 6.3 Principales Ataques a Aplicaciones Web

Un ataque es el conjunto de acciones para comprometer la seguridad de una aplicación. Objetivos: acceso no autorizado, robo o modificación de datos, interrupción del servicio, beneficio económico, instalar software malicioso.

**SQL Injection**

**XSS**

**CSRF**

**Session Hijacking**

**Fuerza bruta**

**Subida de archivos**

**DoS / DDoS**

# 6.3.1 Inyección SQL (SQL Injection)

Introducir instrucciones SQL dentro de los datos ingresados por un usuario para modificar el comportamiento de una consulta sobre la base de datos. Ocurre cuando la aplicación construye consultas concatenando datos del usuario sin validarlos.

## Ejemplo simplificado

Consulta construida concatenando los datos ingresados por el usuario

```
String sql = "SELECT * FROM usuarios " + "WHERE usuario = '" + usuario + "' AND contraseña = '" + password + "'";
```

Consulta esperada:

```
SELECT * FROM usuarios WHERE usuario='juan' AND contraseña='123456';
```

Atacante ingresa contraseña: `' OR '1'='1`

→ La condición '1'='1' siempre es verdadera → acceso sin conocer la contraseña real.
El problema no está en SQL, sino en cómo la aplicación construye la consulta.

Este tipo de ataque afecta principalmente a la base de datos.

# SQL Injection – Consecuencias y prevención

## Consecuencias

- Acceso a información confidencial
- Modificación o eliminación de registros
- Evitar autenticación
- Comprometer integridad de datos

## Medidas preventivas

- Consultas parametrizadas (Prepared Statements)
- Validar todos los datos de entrada
- No concatenar cadenas en SQL
- Principio de mínimo privilegio
- Auditoría de operaciones críticas

Prepared Statements: separan la instrucción SQL de los datos del usuario.

SELECT * FROM usuarios WHERE usuario = ? AND contraseña = ?;

Los datos se interpretan solo como información, no como parte de la consulta.

# 6.3.2 Cross Site Scripting (XSS)

Permite a un atacante introducir código ejecutable en una página web para que sea ejecutado por el navegador de otros usuarios. Afecta principalmente al navegador de la víctima (no a la BD como SQL Injection).

| | |
|---|---|
| **Stored XSS** | Código guardado en la aplicación (comentarios, foros). Afecta a múltiples usuarios. Más peligroso. |
| **Reflected XSS** | El código se envía en la solicitud y la aplicación lo devuelve en la respuesta. Suele usarse con enlaces fraudulentos. |
| **DOM-based** | La vulnerabilidad se produce en el navegador mediante scripts del lado cliente. |

¿Como ocurre?
El usuario escribe un mensaje y la aplicación lo muestra posteriormente a otros. Si la aplicación no valida el contenido ingresado, un atacante podría intentar enviar código ejecutable.

Ejemplo simplificado:

```
<script>alert('XSS')</script>
```

Cuando otro usuario visite la página, el navegador interpretará ese contenido como código y lo ejecutará.

El problema aparece porque la aplicación:

- acepta contenido ingresado por el usuario;
- lo almacena o lo muestra directamente;
- no lo convierte a texto seguro antes de enviarlo al navegador.

# XSS – Objetivos y consecuencias

- Robar cookies o sesiones
- Redirigir a sitios fraudulentos
- Mostrar contenido falso
- Ejecutar acciones en nombre del usuario
- Obtener información del navegador

Las principales medidas para prevenir XSS son:

- validar las entradas del usuario;
- codificar o escapar el contenido antes de mostrarlo;
- evitar insertar directamente datos del usuario en el HTML;
- utilizar frameworks que incluyan protección automática;

Ejemplo

Supóngase una aplicación de foros. El atacante publica un comentario que contiene código malicioso. La aplicación guarda el comentario y luego lo muestra a todos los usuarios.
El flujo seria:
1. Atacante publica comentario malicioso
2. Aplicación lo almacena sin validación
3. Otro usuario visita la página
4. El navegador ejecuta el código
5. Se compromete la sesión o la información

La vulnerabilidad no está en el navegador, sino en la aplicación que permitió almacenar y mostrar contenido no seguro.

# 6.3.3 Cross Site Request Forgery (CSRF) – Falsificacion de peticiones entre sitios

Es un ataque mediante el cual un atacante consigue que un usuario autenticado ejecute acciones no deseadas sobre una aplicación web sin ser consciente de ello.
A diferencia del ataque XSS, donde el navegador ejecuta código malicioso, en un ataque CSRF el navegador envía solicitudes legítimas al servidor utilizando la sesión ya iniciada por el usuario. Esto ocurre porque la aplicación confía en que toda solicitud proveniente del navegador del usuario es legítima.

## ¿Por qué ocurre?

El ataque es posible cuando la aplicación:

- Identifica al usuario únicamente mediante la sesión activa;
- No verifica el origen de las solicitudes;
- No incorpora mecanismos adicionales para validar que la petición fue generada desde la propia aplicación.

## Prevención principal

- Tokens anti-CSRF en formularios y solicitudes sensibles
- Verificar el origen (Origin / Referer) de las peticiones
- Usar el atributo SameSite en cookies de sesión
- Requerir reautenticación para operaciones críticas

# 6.3.4 Robo de Sesiones (Session Hijacking)

Consiste en obtener o utilizar el identificador de sesión de un usuario legítimo para actuar en su nombre. El identificador de sesión es un elemento sensible: quien lo posea puede ser tratado como el usuario autenticado.

## ¿Cómo funciona una sesión web?

1. El usuario introduce sus credenciales.
2. La aplicación verifica la identidad.
3. El servidor crea una sesión.
4. El servidor entrega al navegador un identificador de sesión.
5. El navegador conserva dicho identificador, normalmente mediante una cookie.
6. En las solicitudes posteriores, el navegador envía el identificador.
7. El servidor utiliza ese identificador para reconocer al usuario.

## ¿Cómo puede producirse el robo de una sesión?

- transmisión de información sin utilizar HTTPS;
- vulnerabilidades XSS;
- malware instalado en el equipo del usuario;
- almacenamiento inseguro de cookies;
- utilización de equipos públicos o compartidos;
- configuraciones incorrectas de las sesiones.

# 6.3.4.5 Robo de Sesiones – Ejemplo y medidas preventivas

Un usuario inicia sesión en una aplicación web desde una computadora pública. La aplicación mantiene la sesión activa mediante una cookie. El usuario abandona el equipo sin cerrar correctamente la sesión.
Posteriormente, otra persona utiliza el mismo equipo y accede al navegador. Si la sesión continua activa, el segundo usuario podría acceder a la aplicación utilizando la sesión del usuario anterior. En este caso no fue necesario conocer la contraseña: se aprovechó una sesión que permanecía activa.

- Usar HTTPS (no transmitir el ID en claro)
- Identificadores aleatorios y difíciles de predecir
- Renovar el ID después de autenticarse
- Expiración por inactividad
- Invalidar sesión al cerrar sesión
- Cookies Secure, HttpOnly y SameSite

# 6.3.5 Ataques de Fuerza Bruta - Variantes

Intentar descubrir credenciales probando sistemáticamente muchas combinaciones de usuario y contraseña. Puede automatizarse con herramientas.

## Existen diferentes variantes que conviene distinguir

**Fuerza bruta tradicional**
Se prueban sistemáticamente diferentes combinaciones de caracteres hasta encontrar la contraseña correcta. Es un método que puede requerir una cantidad muy elevada de intentos.

**Ataque de diccionario**
En lugar de probar todas las combinaciones posibles, se utiliza una lista de contraseñas o palabras frecuentes. Por ejemplo: 123456, password, qwerty, admin, 12345678

**Credential Stuffing**
El atacante utiliza combinaciones de usuario y contraseña obtenidas previamente de filtraciones de datos para intentar acceder a otros servicios. Por ejemplo, si una persona utiliza la misma contraseña en varios sitios web y una de esas plataformas sufre una filtración

# 6.3.5.4 Ataques de Fuerza Bruta – Medidas preventivas

Las principales medidas para reducir el riesgo de ataques de fuerza bruta son:

- Límite de intentos de acceso (account lockout)
- Captcha o desafíos tras varios fallos
- Autenticación multifactor (MFA)
- Contraseñas fuertes y políticas de complejidad
- Monitoreo de intentos fallidos
- Retrasos progresivos entre intentos

# 6.3.6 Subida de Archivos Maliciosos – Concepto

Muchas aplicaciones permiten subir archivos (fotos de perfil, documentos, comprobantes, currículums, adjuntos). Esta funcionalidad es un riesgo cuando no hay controles adecuados.

Un atacante puede subir un archivo preparado para que el servidor lo almacene, lo ejecute o lo utilice de forma no prevista.

## ¿Cómo ocurre el ataque?

La aplicación espera una imagen (foto.jpg). El atacante envía un archivo con código ejecutable (archivo_malicioso.php). Si solo se verifica el nombre o la extensión de forma incorrecta y se almacena en una ubicación ejecutable, se produce el incidente.
El problema no es permitir subir archivos, sino no controlar qué se almacena y cómo lo trata el servidor.

# 6.3.6 Subida de Archivos – Consecuencias y prevención

## Consecuencias posibles

- Almacenamiento de archivos no autorizados
- Modificación de información
- Instalación de malware
- Ejecución de código malicioso
- Acceso no autorizado al servidor
- Compromiso de otros componentes

## Medidas preventivas

- Limitar tipos de archivos permitidos (lista blanca)
- Establecer tamaño máximo
- Usar nombres generados por la aplicación
- Verificar el contenido real, no solo la extensión
- Almacenar fuera de ubicaciones ejecutables
- Controles de permisos y análisis de malware

# 6.3.6 Subida de Archivos – Punto clave

## Importante

No basta con comprobar que el nombre del archivo termine en una extensión permitida.

Ejemplo: fotografia.jpg no garantiza que el archivo sea realmente una imagen.

La aplicación debe realizar controles adicionales para determinar si el contenido corresponde al tipo permitido.

Si algún control falla, el archivo debe ser rechazado.

# 6.3.7 Denegación de Servicio (DoS) – Concepto

Un ataque de Denegación de Servicio (DoS – Denial of Service) tiene como objetivo impedir o dificultar que los usuarios legítimos puedan acceder a una aplicación o servicio.

A diferencia de otros ataques (robo o modificación de datos), el objetivo principal es afectar la disponibilidad del servicio.

## ¿Cómo funciona?

El atacante genera una cantidad excesiva de solicitudes o tráfico hacia el servidor. Si el servidor recibe más de lo que puede procesar, se agotan recursos (CPU, memoria, ancho de banda, conexiones).

Cuando los recursos se saturan, los usuarios legítimos no pueden acceder o el servicio se vuelve extremadamente lento.

# 6.3.7 DoS vs DDoS

## DoS

Denial of Service

El ataque se origina generalmente desde un único equipo o una única fuente.

Más sencillo de identificar y, en muchos casos, de mitigar (bloqueo de IP de origen).

## DDoS

Distributed Denial of Service

Las solicitudes provienen de numerosos equipos distribuidos (botnet: dispositivos previamente comprometidos).

Más difícil de mitigar por la distribución del tráfico.

# 6.3.7 DoS/DDoS – Prevención y relación con la disponibilidad

## Medidas preventivas (combinar varios mecanismos)

- Monitoreo del tráfico de red
- Limitación de solicitudes (rate limiting)
- Sistemas de filtrado
- Balanceadores de carga
- Servicios especializados de mitigación DDoS
- Distribución de la infraestructura
- Planes de contingencia para mantener el servicio

Relación con la disponibilidad (CIA): una aplicación puede tener datos confidenciales e íntegros, pero si los usuarios no pueden acceder, existe un problema de seguridad. No basta con proteger el robo o la modificación de datos.

# 6.4 OWASP Top 10

OWASP (Open Worldwide Application Security Project) es una organización sin fines de lucro dedicada a mejorar la seguridad del software. El OWASP Top 10 es un documento de referencia que enumera los riesgos de seguridad más críticos en aplicaciones web.

## Importancia

- Referencia estándar para desarrolladores y auditores
- Ayuda a priorizar controles de seguridad
- Se actualiza periódicamente según riesgos observados
- Base para checklists y pruebas de seguridad

# 6.4.3 Principales categorías del OWASP Top 10

| Nº | Categoría | Descripción general |
|---|---|---|
| 1 | A01 – Broken Access Control | Problemas que permiten acceder a recursos o realizar acciones que el usuario no debería poder realizar. |
| 2 | A02 – Cryptographic Failures | Fallos relacionados con la protección criptográfica de información sensible. |
| 3 | A03 – Injection | Entrada de datos que permite modificar la interpretación de instrucciones por parte de una aplicación. Incluye SQL Injection. |
| 4 | A04 – Insecure Design | Problemas originados por decisiones o diseños que no contemplan adecuadamente la seguridad. |
| 5 | A05 – Security Misconfiguration | Configuraciones incorrectas o inseguras de aplicaciones, servidores o componentes. |
| 6 | A06 – Vulnerable and Outdated Components | Utilización de componentes con vulnerabilidades conocidas o versiones obsoletas. |
| 7 | A07 – Identification and Authentication Failures | Problemas relacionados con la identificación y autenticación de usuarios. |
| 8 | A08 – Software and Data Integrity Failures | Problemas relacionados con la falta de protección de la integridad del software o de los datos. |
| 9 | A09 – Security Logging and Monitoring Failures | Falta de registros, monitoreo o mecanismos adecuados para detectar actividades sospechosas. |
| 10 | A10 – Server-Side Request Forgery (SSRF) | Vulnerabilidades que permiten que una aplicación realice solicitudes no autorizadas hacia otros recursos. |

# 6.5 Autenticación y Gestión de Sesiones

## Autenticación

Verifica la identidad del usuario antes de permitir el acceso a recursos. Determina quién es el usuario.

## Autorización

Determina qué recursos y operaciones puede utilizar ese usuario una vez autenticado.

## Gestión de sesiones (HTTP no mantiene estado)

Una aplicación web necesita un mecanismo que permita recordar que diferentes solicitudes pertenecen al mismo usuario. La gestión de sesiones permite mantener esta asociación

1. Usuario se autentica → 2. Servidor verifica credenciales → 3. Crea sesión → 4. Genera ID de sesión → 5. Envía al navegador → 6. Navegador lo usa en peticiones posteriores → 7. Servidor reconoce al usuario.

# 6.5.3 Protección de las Sesiones

### HTTPS

Comunicación cifrada; no transmitir el ID en claro

### ID seguros

Aleatorios y difíciles de predecir (no valores simples)

### Renovación

Cambiar el ID después de autenticarse

### Expiración

Tiempo límite de inactividad

### Invalidación

Al cerrar sesión, el ID debe dejar de ser válido

### Cookies seguras

Secure, HttpOnly, SameSite

# 6.6.1 Firewall Tradicional

Mecanismo que controla el tráfico de red que entra o sale de un sistema o red, aplicando reglas (IP, puertos, protocolos). Protege principalmente la infraestructura de red.

**Limitación frente a aplicaciones web**

No está diseñado para analizar en profundidad el contenido de las solicitudes HTTP/HTTPS. Una petición con SQL Injection puede parecer tráfico válido (puerto y protocolo correctos) y ser permitida.

Idea clave: permitir una comunicación de red no significa que el contenido sea seguro para la aplicación.

# 6.6.2 Web Application Firewall (WAF)

Mecanismo específico para proteger aplicaciones web. Analiza las solicitudes dirigidas a la aplicación e identifica patrones asociados a ataques (SQL Injection, XSS, etc.). Actúa como capa delante de la aplicación.

## Flujo simplificado

Usuario → WAF (analiza) → permite o bloquea → si es legítima → Aplicación

- ✓ Capa adicional de protección
- ✓ Detecta patrones de ataque
- ✓ Bloquea solicitudes sospechosas
- ✓ Reglas específicas por aplicación

- ✗ No reemplaza desarrollo seguro
- ✗ Puede generar falsos positivos
- ✗ No detecta todas las vulnerabilidades
- ✗ Requiere mantenimiento

# 6.6.3 HTTPS y Certificados Digitales

| HTTP | HTTPS |
| --- | --- |
| Sin protección criptográfica de transporte | Comunicación web protegida |
| Datos pueden quedar expuestos en tránsito | Datos protegidos durante el tránsito |
| No verifica identidad del servidor con certificados | Utiliza certificados digitales |
| Inadecuado para información sensible | Adecuado para información sensible |

HTTPS protege la comunicación, pero no corrige vulnerabilidades de la aplicación (SQL Injection, XSS, etc.).

# Certificado Digital y Autoridad Certificadora

Certificado digital: documento electrónico que asocia una identidad con una clave usada en la comunicación segura. El navegador verifica dominio, validez, emisor y cadena de confianza.

Autoridad Certificadora (CA): entidad que emite certificados y forma parte del sistema de confianza de HTTPS.

## El candado del navegador

Indica que la conexión usa HTTPS correctamente. No significa que el sitio sea completamente seguro ni legítimo en todos los aspectos.
Un sitio puede tener certificado válido y aún así tener SQL Injection u otras vulnerabilidades.
HTTPS protege la comunicación; no corrige las vulnerabilidades de la aplicación.

# 6.7 Buenas Prácticas para la Seguridad en Aplicaciones Web

### Validar entradas

Evitar datos manipulados procesados de forma insegura

### Consultas parametrizadas

Prevenir SQL Injection

### Controles de acceso

Evitar acceso a recursos no autorizados

### Gestionar sesiones

Reducir robo, fijación o uso indebido

### Usar HTTPS

Proteger información en tránsito

### Controlar archivos subidos

Reducir riesgo de archivos maliciosos

### Controlar solicitudes

Mitigar impacto de DoS

### WAF cuando corresponda

Capa adicional frente a ataques web

# 6.7.2 Principios fundamentales

1. No confiar directamente en los datos proporcionados por el usuario
2. Validar y controlar las entradas antes de procesarlas
3. Separar los datos de las instrucciones (Prepared Statements)
4. Verificar siempre los permisos del usuario antes de permitir una operación
5. Proteger las sesiones después de la autenticación
6. Proteger las comunicaciones mediante HTTPS
7. Agregar capas de protección (WAF) cuando corresponda