Facultad de Ingeniería

Unidad 6. Seguridad en aplicaciones web

Apunte de cátedra

Contenido

6. Seguridad en Aplicaciones Web
6.1. Introducción a la Seguridad en Aplicaciones Web

A diferencia de las aplicaciones tradicionales instaladas en una computadora, las aplicaciones web se ejecutan sobre servidores conectados a Internet y pueden ser utilizadas simultáneamente por miles o incluso millones de usuarios desde cualquier parte del mundo.

Esta característica representa una gran ventaja desde el punto de vista de la disponibilidad y accesibilidad, pero también incrementa considerablemente la superficie de ataque. Al estar permanentemente expuestas a Internet, estas aplicaciones son uno de los objetivos preferidos de los ciberdelincuentes.

La Seguridad en Aplicaciones Web tiene como finalidad **proteger tanto la aplicación como la información que procesa frente a accesos no autorizados, modificaciones indebidas, robo de información o interrupciones del servicio**.

Una falla de seguridad en una aplicación web puede ocasionar importantes consecuencias para una organización, entre ellas:

- pérdida de información confidencial;
- robo de datos personales de los usuarios;
- fraude económico;
- interrupción del servicio;
- pérdida de confianza de clientes y usuarios;
- sanciones legales derivadas del incumplimiento de normativas de protección de datos.

Por estas razones, la seguridad debe considerarse desde las primeras etapas del desarrollo de una aplicación y mantenerse durante todo su ciclo de vida.

6.1.1. Importancia de la Seguridad en Aplicaciones Web

Entre las principales razones por las cuales las aplicaciones web requieren mecanismos de seguridad adecuados se encuentran:

- están disponibles las 24 horas del día;
- pueden ser accedidas desde cualquier lugar del mundo;
- reciben solicitudes provenientes de usuarios desconocidos;
- procesan grandes volúmenes de información sensible;
- suelen integrarse con otros sistemas y bases de datos.

La combinación de estos factores convierte a las aplicaciones web en uno de los objetivos más frecuentes de los ataques informáticos.

6.1.2. Objetivos de la Seguridad en Aplicaciones Web

Los mecanismos de seguridad implementados en una aplicación web persiguen los mismos objetivos fundamentales estudiados en la Unidad 1 de esta asignatura: confidencialidad, integridad, disponibilidad.

6.2. Arquitectura Cliente-Servidor de una Aplicación Web

Para comprender cómo se producen los principales ataques contra una aplicación web es necesario conocer, al menos de manera general, cómo se encuentra organizada su arquitectura.

Las aplicaciones web modernas utilizan el modelo **cliente-servidor**, en el cual las tareas se distribuyen entre dos componentes principales:

- el **cliente**, que solicita un servicio;
- el **servidor**, que procesa la solicitud y devuelve una respuesta.

Este modelo permite que numerosos usuarios puedan acceder simultáneamente a una misma aplicación utilizando únicamente un navegador web, sin necesidad de instalar programas específicos en sus computadoras.

Desde el punto de vista de la seguridad, cada uno de los componentes que participan en esta arquitectura puede presentar vulnerabilidades que, si son explotadas por un atacante, pueden comprometer la información almacenada por la organización.

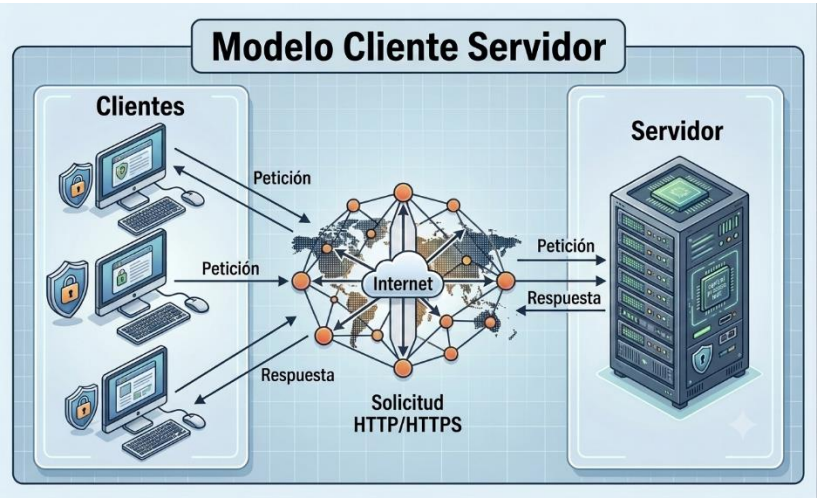
Por este motivo, resulta importante conocer el recorrido que realiza una solicitud desde que un usuario accede a una página web hasta que recibe la respuesta correspondiente.

6.2.1. Arquitectura Cliente-Servidor

La arquitectura cliente-servidor es un modelo de comunicación en el cual un programa denominado **cliente** solicita información o servicios a otro programa denominado **servidor**.

En una aplicación web, el cliente suele ser un navegador, mientras que el servidor ejecuta la aplicación y administra la información almacenada en la base de datos. El cliente no realiza la mayor parte del procesamiento de la información, sino que envía solicitudes al servidor y presenta al usuario las respuestas recibidas.

Actualmente, la gran mayoría de las aplicaciones utilizadas por empresas, organismos públicos y universidades emplean este modelo de funcionamiento.



6.2.2. Componentes Principales de una Aplicación Web

Aunque desde la perspectiva del usuario una aplicación web parece ser un único sistema, internamente está compuesta por diversos elementos que trabajan de forma coordinada.

Los componentes más importantes son:

- navegador web;
- servidor web;
- servidor de aplicaciones;
- base de datos.

Cada uno cumple una función específica dentro del procesamiento de la información.

6.2.2.1. Navegador Web (Cliente)

Constituye la interfaz mediante la cual el usuario interactúa con la aplicación.

Su función consiste en:

- enviar solicitudes al servidor;
- recibir las respuestas;
- mostrar la información al usuario.

Desde el punto de vista de la seguridad, el navegador puede ser víctima de ataques como:

- robo de sesiones;
- phishing;
- Cross Site Scripting (XSS).

6.2.2.2. Servidor Web

Recibe las solicitudes provenientes del navegador y las dirige hacia la aplicación correspondiente.

Entre sus funciones principales se encuentran:

- aceptar conexiones de los usuarios;
- entregar páginas web;
- administrar recursos estáticos como imágenes y documentos;
- comunicarse con el servidor de aplicaciones.

Ejemplos de servidores web ampliamente utilizados son:

- Apache HTTP Server;
- Nginx;
- Microsoft IIS.

6.2.2.3. Servidor de Aplicaciones

Tiene como función principal proporcionar un entorno para ejecutar, gestionar y desplegar aplicaciones dinámicas y su lógica de negocio.

Es el encargado de:

- validar la información ingresada por los usuarios;
- aplicar las reglas de funcionamiento del sistema;
- consultar la base de datos;
- generar la respuesta que será enviada al navegador.

Por ejemplo, cuando un estudiante solicita visualizar sus notas, el servidor de aplicaciones verifica su identidad, consulta la información correspondiente y genera la página que finalmente visualizará el usuario.

Ejemplo de servidores de aplicaciones:

- Apache Tomcat (Java),
- WildFly/JBoss,
- WebLogic,
- WebSphere,
- Gunicorn o uWSGI (Python),
- PM2 (Node.js).

6.2.2.4. Base de Datos

La base de datos almacena toda la información utilizada por la aplicación.

Dependiendo del sistema, puede contener:

- usuarios;
- contraseñas cifradas;
- información académica;
- clientes;
- productos;
- facturas;
- operaciones financieras.

La protección de esta información constituye uno de los principales objetivos de la seguridad informática. Si un atacante logra acceder o modificar la base de datos sin autorización, las consecuencias pueden ser extremadamente graves para la organización.

6.2.3. Flujo de una Petición Web

Cada vez que un usuario interactúa con una aplicación web, se produce un intercambio de información entre el navegador y el servidor. Este proceso, denominado **petición web**, ocurre de forma transparente para el usuario y suele completarse en pocos segundos.

Comprender este flujo permite identificar en qué etapa pueden producirse vulnerabilidades o ataques.

El proceso básico es el siguiente:

Paso 1. El usuario realiza una acción

El usuario ingresa una dirección web, completa un formulario, hace clic en un botón o solicita determinada información desde el navegador.

Paso 2. El navegador envía la solicitud

El navegador envía una petición al servidor web utilizando el protocolo HTTP o HTTPS.

La solicitud puede incluir información como:

- nombre de usuario;
- contraseña;
- datos de un formulario;
- parámetros de búsqueda;
- archivos enviados por el usuario.

Paso 3. El servidor recibe la petición

El servidor web recibe la solicitud y la deriva al servidor de aplicaciones para que sea procesada.

Paso 4. La aplicación procesa la información

El servidor de aplicaciones analiza la petición recibida.

Dependiendo de la operación solicitada puede:

- validar los datos ingresados;
- verificar la identidad del usuario;
- comprobar permisos de acceso;
- consultar o modificar información almacenada.

Paso 5. Acceso a la base de datos

Si la operación requiere consultar información, el servidor de aplicaciones accede a la base de datos. En esta etapa se almacenan o recuperan los datos solicitados.

Paso 6. Generación de la respuesta

Una vez procesada la solicitud, el servidor genera la respuesta correspondiente.

Por ejemplo:

- una página web;
- una lista de productos;
- una factura electrónica;
- las calificaciones de un estudiante.

Paso 7. El navegador muestra el resultado

Finalmente, el navegador presenta la información al usuario. Todo este proceso suele realizarse en fracciones de segundo.

Ejemplo práctico

Supóngase que un cliente desea consultar el saldo de su cuenta bancaria.

El proceso ocurre de la siguiente manera:

1. El cliente ingresa al sitio del banco.
2. Introduce su usuario y contraseña.
3. El navegador envía la solicitud al servidor.
4. El servidor verifica las credenciales.
5. Consulta la base de datos.
6. Recupera el saldo correspondiente.

7. Envía la respuesta.
8. El navegador muestra el saldo en pantalla.

Si durante alguna de estas etapas existe una vulnerabilidad, un atacante podría intentar aprovecharla para acceder a información confidencial o alterar el funcionamiento de la aplicación.

6.2.4. Riesgos asociados a cada componente

Cada componente de una aplicación web cumple una función específica y presenta riesgos particulares. Comprender estas amenazas permite implementar medidas de seguridad adecuadas para reducir la probabilidad de sufrir un incidente.

6.2.4.1. Navegador Web

El navegador constituye el punto de interacción entre el usuario y la aplicación.

Algunos riesgos asociados son:

- robo de sesiones;
- ejecución de código malicioso mediante XSS;
- phishing;
- almacenamiento inseguro de credenciales.

6.2.4.2. Servidor Web

El servidor web recibe las solicitudes provenientes de Internet.

Entre sus principales riesgos pueden mencionarse:

- configuraciones incorrectas;
- software desactualizado;
- exposición innecesaria de servicios;
- errores en la configuración de seguridad.

6.2.4.3. Servidor de Aplicaciones

Es el componente donde se ejecuta la lógica de negocio.

Las principales vulnerabilidades suelen originarse aquí.

Ejemplos:

- validación incorrecta de datos;
- errores de programación;
- control de acceso insuficiente;
- manejo incorrecto de sesiones.

6.2.4.4. Base de Datos

La base de datos almacena la información crítica de la organización.

Los riesgos más importantes son:

- acceso no autorizado;
- modificación de registros;
- eliminación de información;
- robo de datos personales;
- pérdida de integridad de la información.

Una vulnerabilidad en este componente puede afectar a toda la organización.

En la siguiente tabla se muestran los riesgos asociados a cada componente

Componente	Principales riesgos
Navegador	Phishing, XSS, robo de sesiones
Servidor Web	Configuración incorrecta, software desactualizado
Servidor de Aplicaciones	SQL Injection, XSS, CSRF, errores de programación
Base de Datos	Robo, modificación o eliminación de información

Puede observarse que la seguridad de una aplicación web no depende únicamente del servidor o de la base de datos. Todos los componentes deben protegerse de manera conjunta para garantizar la confidencialidad, integridad y disponibilidad de la información.

6.3. Principales Ataques a Aplicaciones Web

Las aplicaciones web se encuentran expuestas a numerosos ataques que buscan explotar vulnerabilidades presentes en el software, la configuración del servidor o los mecanismos de autenticación.

Un **ataque** consiste en el conjunto de acciones realizadas por un atacante con el objetivo de comprometer la seguridad de una aplicación informática.

Generalmente, los ataques buscan alguno de los siguientes objetivos:

- obtener acceso no autorizado;
- robar información;
- modificar datos;
- interrumpir el funcionamiento del sistema;
- obtener beneficios económicos;
- instalar software malicioso.

La mayoría de estos ataques aprovecha errores de programación, configuraciones incorrectas o controles de seguridad insuficientes.

En los apartados siguientes se estudiarán algunos de los ataques más frecuentes contra aplicaciones web.

6.3.1. Inyección SQL (SQL Injection)

La **Inyección SQL (SQL Injection)** es uno de los ataques más conocidos contra aplicaciones web y consiste en introducir instrucciones SQL dentro de los datos ingresados por un usuario con el objetivo de modificar el comportamiento de una consulta realizada sobre una base de datos.

Este ataque ocurre cuando una aplicación construye consultas SQL utilizando directamente los datos ingresados por el usuario, sin validarlos adecuadamente. Como consecuencia, un atacante puede acceder a información que no debería visualizar, modificar registros o incluso eliminar datos almacenados en la base de datos.

6.3.1.1. ¿Cómo ocurre una Inyección SQL?

Supóngase una aplicación web que permite iniciar sesión mediante un nombre de usuario y una contraseña. Cuando un usuario ingresa sus credenciales, la aplicación consulta la base de datos para verificar si existe un registro con esos datos.

La consulta SQL esperada sería la siguiente:

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

Si los datos son correctos, el usuario podrá acceder al sistema. Sin embargo, algunas aplicaciones construyen esta consulta concatenando directamente los datos ingresados por el usuario.

Por ejemplo (pseudocódigo):

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

Mientras el usuario introduzca información válida, el funcionamiento será correcto.

Por ejemplo:

Usuario: juan
Contraseña: 123456

La consulta generada será:

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

Pero un atacante podría ingresar un texto especialmente preparado.

Por ejemplo:

Usuario: juan
Contraseña: ' OR '1'='1

La aplicación construiría la siguiente consulta:

```
SELECT *
FROM usuarios
WHERE usuario = 'juan'
AND contraseña = '' OR '1'='1';
```

Como la condición `'1'='1'` siempre es verdadera, la consulta puede devolver registros que no deberían mostrarse, permitiendo el acceso al sistema sin conocer la contraseña real. Es importante destacar que el problema **no se encuentra en SQL**, sino en la forma incorrecta en que la aplicación construye la consulta.

6.3.1.2. Consecuencias

Una Inyección SQL puede permitir:

- acceder a información confidencial;
- modificar registros almacenados en la base de datos;
- eliminar información importante;
- evitar el proceso de autenticación;
- comprometer la integridad de los datos.

Dependiendo de los privilegios con los que trabaja la aplicación, las consecuencias pueden afectar a toda la organización.

6.3.1.3. Medidas Preventivas

Las principales medidas para prevenir este tipo de ataque son:

- utilizar consultas parametrizadas (*Prepared Statements*);
- validar adecuadamente todos los datos ingresados por el usuario;
- evitar construir consultas SQL concatenando cadenas de texto;
- aplicar el principio del mínimo privilegio a las cuentas utilizadas por la aplicación;
- registrar las operaciones críticas mediante mecanismos de auditoría.

6.3.1.4. ¿Qué son las consultas parametrizadas (Prepared Statements)?

Las **consultas parametrizadas**, también conocidas como **Prepared Statements**, constituyen uno de los mecanismos más eficaces para prevenir ataques de Inyección SQL. A diferencia de una consulta construida concatenando cadenas de texto, una consulta parametrizada separa claramente **la instrucción SQL** de **los datos ingresados por el usuario**.

En lugar de incorporar directamente los valores dentro de la consulta:

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

la aplicación utiliza una consulta con parámetros:

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

Posteriormente, el programa reemplaza cada parámetro (?) por el valor correspondiente utilizando funciones específicas del lenguaje de programación. De esta forma, la base de datos interpreta los datos ingresados únicamente como **información** y no como parte de la instrucción SQL, impidiendo que un usuario modifique la estructura de la consulta.

Por este motivo, el uso de **Prepared Statements** es una de las principales recomendaciones de seguridad propuestas por **OWASP** para prevenir este tipo de vulnerabilidades.

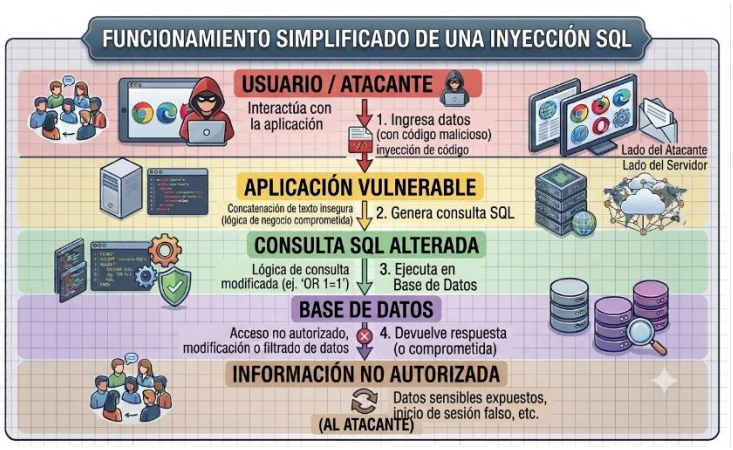
Comparación entre una consulta vulnerable y una consulta parametrizada

Consulta vulnerable	Consulta parametrizada
Construye la consulta concatenando texto.	Utiliza parámetros (?) para separar la consulta de los datos.
El usuario puede alterar la consulta SQL.	Los datos nunca modifican la estructura de la consulta.
Puede producir una Inyección SQL.	Previene la Inyección SQL.

Ejemplo práctico

Una empresa dispone de un sistema web para consultar información de clientes. El formulario solicita únicamente el número de documento. Debido a una programación incorrecta, la aplicación incorpora directamente el valor ingresado por el usuario dentro de la consulta SQL.

Un atacante introduce información especialmente preparada y consigue modificar la consulta realizada por la aplicación, obteniendo acceso a registros pertenecientes a otros clientes. Este incidente podría haberse evitado utilizando consultas parametrizadas y validando adecuadamente los datos recibidos.



6.3.2. Cross Site Scripting (XSS)

El Cross Site Scripting (XSS) es una vulnerabilidad que permite a un atacante introducir código ejecutable dentro de una página web para que dicho código sea ejecutado por el navegador de otros usuarios.

A diferencia de la Inyección SQL, que afecta principalmente a la base de datos, el XSS afecta principalmente al navegador de la víctima.

El objetivo del atacante suele ser:

- robar cookies o sesiones;
- redirigir al usuario a sitios fraudulentos;
- mostrar contenido falso;
- ejecutar acciones en nombre del usuario;
- obtener información del navegador.

La vulnerabilidad aparece cuando una aplicación web muestra información proporcionada por un usuario sin validarla ni codificarla adecuadamente.

6.3.2.1. ¿Cómo ocurre un ataque XSS?

Supóngase una aplicación que permite publicar comentarios. El usuario escribe un mensaje y la aplicación lo muestra posteriormente a otros visitantes.

Comentario normal:

```
Muy buen producto
```

La página mostrará el comentario correctamente. Sin embargo, 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. En este ejemplo solo aparece un mensaje, pero en un ataque real podrían realizarse acciones mucho más peligrosas.

6.3.2.2. ¿Por qué ocurre?

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.

En otras palabras, el navegador no distingue entre:

- contenido legítimo de la página;
- contenido malicioso introducido por un atacante.

6.3.2.3. Tipos de XSS (visión general)

No es necesario estudiar los detalles técnicos, pero es útil conocer que existen tres variantes principales.

XSS almacenado (Stored XSS)

El código malicioso se guarda en la aplicación (por ejemplo, en comentarios, foros o perfiles de usuario). Es la variante más peligrosa porque afecta a múltiples usuarios.

XSS reflejado (Reflected XSS)

El código se envía en una solicitud y la aplicación lo devuelve inmediatamente en la respuesta. Suele utilizarse junto con enlaces fraudulentos.

XSS basado en DOM

La vulnerabilidad se produce en el navegador mediante scripts del lado cliente.

6.3.2.4. Consecuencias

Un ataque XSS puede permitir:

- robo de cookies de sesión;
- secuestro de la sesión del usuario;
- redirección a sitios falsos;

- modificación visual de la página;
- ejecución de acciones en nombre del usuario autenticado.

Por ejemplo, si un usuario tiene una sesión activa en un sistema bancario, un XSS podría intentar aprovechar esa sesión para realizar operaciones no autorizadas.

6.3.2.5. Ejemplo práctico

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 sería:

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

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

6.3.2.6. Medidas preventivas

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;

¿Qué significa codificar o escapar el contenido?

La idea consiste en convertir caracteres especiales en texto seguro.

Por ejemplo, en lugar de enviar:

```
<script>
```

la aplicación envía:

```
&lt;script&gt;
```

El navegador lo mostrará como texto y no lo ejecutará como código.

Comparación entre contenido vulnerable y contenido seguro

Vulnerable	Seguro
`<script>...</script>`	`&lt;script&gt;...&lt;/script&gt;`
El navegador ejecuta el código.	El navegador lo muestra como texto.

6.3.3. Cross Site Request Forgery (CSRF)

El **Cross Site Request Forgery (CSRF)**, conocido en español como **Falsificación 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.

6.3.3.1. ¿Cómo ocurre un ataque CSRF?

Supóngase que un usuario ha iniciado sesión en el sitio web de su banco. Mientras mantiene abierta la sesión, recibe por correo electrónico un enlace aparentemente inofensivo o visita una página web controlada por un atacante.

Sin que el usuario lo advierta, dicha página envía una solicitud al sitio del banco utilizando la sesión que ya se encontraba autenticada. Como el servidor identifica correctamente al usuario mediante su sesión activa, interpreta la solicitud como si hubiera sido realizada voluntariamente por él. En consecuencia, podría ejecutar acciones que el usuario nunca tuvo intención de realizar.

6.3.3.2. Ejemplo práctico

Imaginemos un sistema web de administración de empleados. Un administrador ha iniciado sesión correctamente y deja abierta la aplicación. Posteriormente visita otra página web. Esa página contiene un formulario oculto que envía automáticamente la siguiente solicitud:

```
https://empresa.com/empleados/eliminar?id=25
```

Como el administrador continúa autenticado, el servidor recibe la solicitud utilizando su sesión activa y elimina el empleado indicado. El administrador nunca hizo clic sobre el botón "Eliminar"; sin embargo, la acción fue ejecutada porque el servidor asumió que la petición provenía del propio usuario.

6.3.3.3. ¿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.

En estas condiciones, cualquier solicitud enviada utilizando la sesión autenticada podría ser aceptada por el servidor.

6.3.3.4. Consecuencias

Un ataque CSRF puede permitir:

- modificar datos personales;
- cambiar contraseñas;
- eliminar información;
- realizar compras no autorizadas;
- efectuar transferencias de dinero;
- modificar configuraciones del sistema.

La gravedad dependerá de los permisos que posea el usuario cuya sesión fue utilizada.

6.3.3.5. Medidas Preventivas

Las principales medidas para prevenir ataques CSRF son:

- utilizar **tokens CSRF** en los formularios;
- verificar el origen de las solicitudes recibidas;
- solicitar nuevamente la contraseña para operaciones críticas;
- implementar autenticación multifactor en operaciones sensibles;
- configurar correctamente las cookies de sesión (por ejemplo, utilizando el atributo **SameSite**).

6.3.3.6. ¿Qué es un Token CSRF?

Un **Token CSRF** es un valor único y aleatorio que la aplicación genera para cada sesión o formulario. Cuando el usuario envía un formulario, dicho token también es enviado al servidor. Antes de ejecutar la operación solicitada, el servidor verifica que el token recibido sea válido. Si el token no existe o no coincide con el esperado, la solicitud es rechazada.

Como un atacante desconoce el valor del token generado para cada usuario, no puede construir solicitudes válidas utilizando únicamente la sesión autenticada.

Comparación entre una aplicación vulnerable y una aplicación protegida contra CSRF

Aplicación vulnerable	Aplicación protegida
Acepta cualquier solicitud enviada por el navegador del usuario autenticado.	Verifica que cada solicitud incluya un token CSRF válido.

Aplicación vulnerable	Aplicación protegida
No comprueba el origen de la petición.	Valida que la solicitud provenga de la propia aplicación.
Puede ejecutar acciones no autorizadas.	Rechaza las solicitudes que no superan la validación.

6.3.4. Robo de Sesiones (Session Hijacking)

El **robo de sesiones (Session Hijacking)** es un ataque mediante el cual un atacante obtiene o utiliza de manera no autorizada el identificador de sesión de un usuario para hacerse pasar por él dentro de una aplicación web.

Cuando un usuario inicia sesión, la aplicación necesita recordar que las diferentes solicitudes realizadas posteriormente pertenecen a ese usuario. Para ello, normalmente utiliza un **identificador de sesión**, que puede almacenarse en una cookie del navegador. Si un atacante consigue obtener ese identificador, puede intentar utilizarlo para acceder a la aplicación como si fuera el usuario legítimo, sin necesidad de conocer su contraseña.

6.3.4.1. ¿Cómo funciona una sesión web?

De manera simplificada, el proceso puede representarse de la siguiente forma:

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.

6.3.4.2. ¿Cómo puede producirse el robo de una sesión?

Existen diferentes situaciones que pueden permitir que un atacante obtenga un identificador de sesión.

Entre ellas se encuentran:

- 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.

Por ejemplo, si una aplicación transmite información de autenticación mediante una conexión insegura, un atacante podría intentar interceptar información relacionada con la

sesión. Por este motivo, el uso de **HTTPS** resulta fundamental para proteger las comunicaciones entre el navegador y el servidor.

6.3.4.3. Ejemplo práctico

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.

6.3.4.4. Consecuencias

El robo de una sesión puede permitir al atacante:

- acceder a información privada;
- consultar o modificar datos;
- realizar operaciones en nombre del usuario;
- utilizar las funciones disponibles para la cuenta comprometida.

La gravedad dependerá de los privilegios que posea el usuario cuya sesión fue comprometida. Por ejemplo, el robo de una sesión perteneciente a un administrador puede tener consecuencias mucho más graves que el robo de una cuenta con permisos limitados.

6.3.4.5. Medidas Preventivas

Las principales medidas para proteger las sesiones son:

- utilizar **HTTPS** en toda la aplicación;
- generar identificadores de sesión aleatorios y difíciles de predecir;
- establecer tiempos de expiración para las sesiones;
- invalidar la sesión cuando el usuario cierre sesión;
- proteger las cookies mediante atributos de seguridad;
- evitar mantener sesiones activas indefinidamente;
- renovar el identificador de sesión después de operaciones importantes, especialmente después de la autenticación.

6.3.4.6. Relación con otros ataques

El robo de sesiones puede estar relacionado con otras vulnerabilidades estudiadas anteriormente.

Por ejemplo:

- un **XSS** puede utilizarse para intentar comprometer información asociada a una sesión;
- una comunicación sin **HTTPS** puede facilitar la exposición de información transmitida;
- una mala gestión de las cookies puede facilitar el uso no autorizado de una sesión;
- un **CSRF** puede aprovechar una sesión legítima para realizar acciones no deseadas.

Por lo tanto, la protección de las sesiones debe abordarse de manera conjunta con los mecanismos de autenticación y protección de las comunicaciones.

Principales medidas de protección de sesiones

Medida	Función
HTTPS	Protege la comunicación entre cliente y servidor
Identificador aleatorio	Dificulta la predicción de sesiones
HttpOnly	Reduce el acceso de scripts a las cookies
Secure	Limita el envío de cookies a HTTPS
SameSite	Reduce determinados ataques entre sitios
Expiración	Limita el tiempo de utilización de una sesión
Cierre de sesión	Invalida la sesión cuando deja de utilizarse

El **robo de sesiones** demuestra que proteger únicamente el nombre de usuario y la contraseña no es suficiente. Una aplicación web también debe proteger adecuadamente las sesiones creadas después de la autenticación.

6.3.5. Ataques de Fuerza Bruta

Un **ataque de fuerza bruta** consiste en intentar acceder a una cuenta probando repetidamente diferentes contraseñas hasta encontrar la correcta.

El atacante no necesita conocer previamente la contraseña. Utiliza un conjunto de posibles combinaciones y las prueba de manera automática.

La eficacia de este tipo de ataque depende principalmente de factores como:

- longitud de la contraseña;
- complejidad de la contraseña;
- existencia de mecanismos de bloqueo;
- cantidad de intentos permitidos;

- utilización de autenticación multifactor.

Los ataques de fuerza bruta pueden dirigirse tanto contra aplicaciones web como contra otros sistemas que requieran autenticación.

6.3.5.1. ¿Cómo funciona?

Supóngase que una aplicación web permite iniciar sesión mediante un usuario y una contraseña.

El atacante conoce el nombre de usuario:

Usuario: juan

Pero desconoce la contraseña. Automáticamente intenta diferentes posibilidades:

123456
123457
123458
password
juan123
...
Si la aplicación permite realizar una cantidad ilimitada de intentos, el atacante puede continuar hasta encontrar la contraseña correcta.

6.3.5.2. Tipos de ataques relacionados

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
```

Este método puede ser más eficiente cuando los usuarios utilizan contraseñas simples o predecibles.

Credential Stuffing

En este caso, 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, el atacante puede intentar utilizar esas mismas credenciales en otros servicios.

6.3.5.3. Consecuencias

Si un ataque tiene éxito, el atacante puede acceder a la cuenta comprometida y utilizar los permisos disponibles para ese usuario.

Las consecuencias pueden incluir:

- acceso a información privada;
- modificación de datos;
- suplantación de identidad;
- realización de operaciones no autorizadas;
- utilización de la cuenta para realizar otros ataques.

El impacto será mayor cuando la cuenta comprometida tenga privilegios administrativos.

6.3.5.4. Medidas Preventivas

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

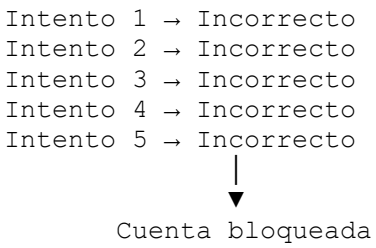
- utilizar contraseñas largas y difíciles de predecir;
- limitar la cantidad de intentos de autenticación;
- implementar bloqueos temporales después de varios intentos fallidos;
- utilizar mecanismos de CAPTCHA cuando resulte necesario;
- implementar autenticación multifactor (MFA);
- detectar y registrar múltiples intentos fallidos;
- utilizar mecanismos de protección contra *Credential Stuffing*.

La autenticación multifactor resulta especialmente importante porque, aunque un atacante consiga descubrir una contraseña, todavía necesitará superar el segundo factor de autenticación.

6.3.5.5. Ejemplo práctico

Supóngase que una aplicación permite realizar **5 intentos de autenticación** consecutivos. Después del quinto intento incorrecto, la cuenta queda temporalmente bloqueada. Un

atacante que intenta realizar un ataque de fuerza bruta encuentra una importante dificultad:



De esta manera, la aplicación limita considerablemente la cantidad de contraseñas que pueden probarse en un período determinado.

Ataques y medidas de protección

Ataque	Medida de protección
Fuerza bruta	Límite de intentos
Diccionario	Contraseñas robustas
Credential Stuffing	Contraseñas únicas + MFA
Múltiples intentos automáticos	Bloqueo, CAPTCHA y monitoreo

6.3.6. Subida de Archivos Maliciosos

Muchas aplicaciones web permiten que los usuarios suban archivos al servidor. Por ejemplo:

- fotografías de perfil;
- documentos;
- comprobantes;
- currículums;
- archivos adjuntos;
- imágenes utilizadas en publicaciones.

Esta funcionalidad puede representar un riesgo de seguridad cuando la aplicación permite subir archivos sin realizar controles adecuados. Un atacante podría intentar subir un archivo especialmente preparado para conseguir que el servidor lo almacene, lo ejecute o lo utilice de una manera no prevista por los desarrolladores.

6.3.6.1. ¿Cómo ocurre el ataque?

Supóngase una aplicación que permite a los usuarios subir una fotografía de perfil. Una aplicación vulnerable podría aceptar cualquier archivo sin comprobar correctamente su contenido. En ese caso, un atacante podría intentar subir un archivo que no sea realmente una imagen, sino un archivo que contenga código ejecutable.

El problema aparece cuando el servidor almacena ese archivo en una ubicación desde la cual puede ser ejecutado.

6.3.6.2. Ejemplo práctico

Supóngase un sistema de gestión de empleados que permite subir fotografías. La aplicación espera recibir:

```
foto.jpg
```

Pero un atacante intenta enviar un archivo diferente, por ejemplo:

```
archivo_malicioso.php
```

Si la aplicación solamente verifica el nombre o la extensión del archivo de manera incorrecta y permite almacenarlo en una ubicación ejecutable, podría producirse un incidente de seguridad. El problema no consiste simplemente en permitir subir archivos, sino en **no controlar adecuadamente qué archivos pueden ser almacenados y cómo serán tratados posteriormente por el servidor**.

6.3.6.3. Consecuencias

Dependiendo de la vulnerabilidad y de la configuración del servidor, un ataque de este tipo puede provocar:

- almacenamiento de archivos no autorizados;
- ejecución de código malicioso;
- modificación de información;
- acceso no autorizado al servidor;
- instalación de malware;
- compromiso de otros componentes de la aplicación.

La gravedad del incidente dependerá principalmente de los permisos con los que se ejecuta la aplicación y de la configuración del servidor.

6.3.6.4. Medidas Preventivas

Para reducir este riesgo se recomienda:

- limitar los tipos de archivos que pueden cargarse;
- verificar el contenido real del archivo y no solamente su extensión;
- establecer un tamaño máximo para los archivos;
- almacenar los archivos en ubicaciones que no permitan su ejecución;
- utilizar nombres generados por la aplicación en lugar de conservar nombres proporcionados por el usuario;
- aplicar controles de permisos adecuados;
- analizar los archivos mediante mecanismos de seguridad cuando corresponda.

6.3.6.5. Importante

No debe considerarse suficiente comprobar únicamente que el nombre del archivo termine en una determinada extensión.

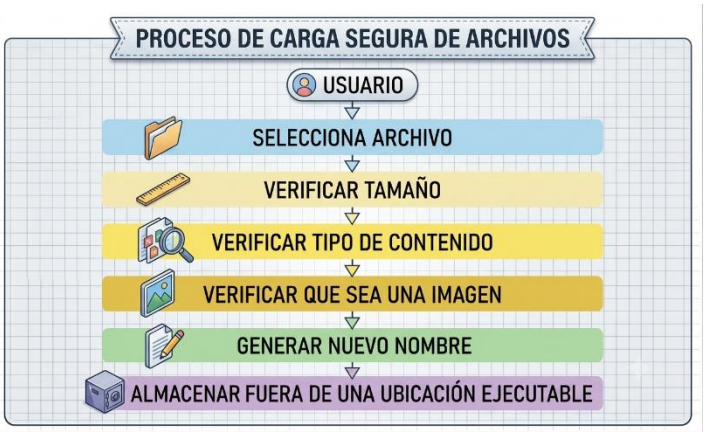
Por ejemplo:

```
fotografia.jpg
```

no garantiza necesariamente que el archivo sea realmente una imagen. La aplicación debe realizar controles adicionales para determinar si el archivo corresponde al tipo de contenido permitido.

6.3.6.6. Ejemplo de una aplicación protegida

Una aplicación para cargar fotografías podría aplicar el siguiente proceso:



Si alguno de los controles falla, la aplicación debería rechazar el archivo.

Controles de seguridad para la subida de archivos

Control	Objetivo
Restricción de tipos	Permitir únicamente archivos necesarios
Límite de tamaño	Evitar archivos excesivamente grandes
Validación del contenido	Comprobar que el archivo corresponde al tipo permitido
Nuevo nombre	Evitar nombres manipulados por el usuario
Ubicación no ejecutable	Impedir la ejecución del archivo
Control de permisos	Limitar las acciones que puede realizar el archivo

6.3.7. Denegación de Servicio (DoS y DDoS)

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 los ataques estudiados anteriormente, cuyo objetivo suele ser obtener información o modificar datos, en un ataque DoS el objetivo principal es afectar la **disponibilidad** del servicio.

6.3.7.1. ¿Cómo funciona un ataque DoS?

En términos generales, el atacante genera una cantidad excesiva de solicitudes o tráfico dirigido hacia un servidor. Si el servidor recibe más solicitudes de las que puede procesar, sus recursos pueden agotarse.

Entre los recursos que pueden verse afectados se encuentran:

- capacidad de procesamiento;
- memoria;
- conexiones disponibles;
- ancho de banda;
- recursos de la aplicación.

Como consecuencia, los usuarios legítimos pueden experimentar lentitud o directamente no poder acceder al servicio.

6.3.7.2. DoS y DDoS

La principal diferencia entre ambos ataques está relacionada con la cantidad de equipos utilizados para generar las solicitudes.

6.3.7.3. DoS – Denial of Service

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

6.3.7.4. DDoS – Distributed Denial of Service

En un ataque **DDoS**, las solicitudes provienen simultáneamente de numerosos equipos distribuidos. Estos equipos pueden pertenecer a una **botnet**, es decir, un conjunto de dispositivos previamente comprometidos que pueden ser controlados por un atacante.

La distribución del tráfico hace que los ataques DDoS sean generalmente más difíciles de mitigar que los ataques provenientes de una única fuente.

6.3.7.5. Ejemplo práctico

Supóngase que una universidad dispone de un sistema web para realizar la inscripción a materias.

En condiciones normales, el servidor recibe solicitudes de cientos de estudiantes. Durante un ataque DDoS, miles de solicitudes pueden ser enviadas simultáneamente. Aunque el sistema continúe funcionando internamente, los estudiantes legítimos pueden no conseguir acceder.

6.3.7.6. Consecuencias

Un ataque DoS o DDoS puede provocar:

- indisponibilidad del servicio;
- lentitud extrema;
- pérdida de operaciones;
- interrupción de actividades comerciales;
- pérdidas económicas;
- afectación de la imagen de la organización.

Por ejemplo, si un sitio de comercio electrónico permanece inaccesible durante varias horas, la organización puede perder ventas y clientes.

6.3.7.7. Medidas Preventivas

La protección contra DoS y DDoS requiere normalmente combinar diferentes mecanismos.

Entre ellos:

- monitoreo del tráfico de red;
- limitación de solicitudes (*rate limiting*);
- utilización de sistemas de filtrado;
- balanceadores de carga;
- servicios especializados de mitigación DDoS;
- distribución de la infraestructura;
- planes de contingencia para mantener el servicio disponible.

No existe una única medida que garantice por sí sola la protección contra todos los ataques DDoS.

6.3.7.8. DoS y DDoS

Característica	DoS	DDoS
Fuentes del ataque	Una o pocas	Numerosas
Distribución	Generalmente centralizada	Distribuida
Objetivo	Afectar la disponibilidad	Afectar la disponibilidad
Dificultad de mitigación	Generalmente menor	Generalmente mayor

6.3.7.9. Relación con la disponibilidad

Los ataques DoS y DDoS permiten observar claramente la importancia del principio de **disponibilidad**. Una aplicación puede mantener sus datos confidenciales y correctamente protegidos, pero si los usuarios legítimos no pueden acceder al servicio, existe igualmente un problema de seguridad.

Por este motivo, la protección de aplicaciones web no debe centrarse únicamente en evitar el robo o modificación de información, sino también en garantizar que los servicios permanezcan disponibles.

6.4. OWASP Top 10

El **OWASP Top 10** es una referencia ampliamente utilizada para identificar y comprender los principales riesgos de seguridad asociados con las aplicaciones web.

El proyecto es desarrollado por **OWASP (Open Worldwide Application Security Project)**, una organización dedicada a mejorar la seguridad del software y de las aplicaciones.

El OWASP Top 10 presenta una clasificación de las categorías de riesgos que deben recibir especial atención durante el desarrollo, implementación y mantenimiento de aplicaciones web.

No constituye una lista exhaustiva de todas las vulnerabilidades existentes, sino una **referencia educativa y de concientización** que permite identificar los riesgos más importantes.

En esta unidad no se desarrollarán individualmente las diez categorías, ya que varias de ellas fueron analizadas previamente mediante los ataques y vulnerabilidades estudiados en el apartado 6.3.

6.4.1. ¿Qué es OWASP?

OWASP (Open Worldwide Application Security Project) es una organización internacional sin fines de lucro dedicada a mejorar la seguridad del software. Su objetivo

es proporcionar conocimientos, herramientas, metodologías y recursos que permitan desarrollar aplicaciones más seguras. OWASP produce numerosos recursos relacionados con la seguridad de aplicaciones, entre ellos:

- guías de seguridad;
- herramientas de análisis;
- documentación técnica;
- metodologías;
- proyectos educativos;
- estándares y buenas prácticas.

Uno de sus recursos más conocidos es el **OWASP Top 10**, utilizado como referencia para identificar los principales riesgos de seguridad en aplicaciones web.

Es importante comprender que OWASP **no es una ley ni una norma obligatoria**. Sus publicaciones constituyen referencias y buenas prácticas que pueden ser utilizadas por desarrolladores, administradores y profesionales de Seguridad Informática.

6.4.2. Importancia del OWASP Top 10

El OWASP Top 10 permite disponer de una visión general de los riesgos más importantes que pueden afectar a una aplicación web.

Su importancia radica principalmente en que:

- facilita la identificación de riesgos;
- ayuda a concientizar a desarrolladores y organizaciones;
- permite establecer prioridades de seguridad;
- sirve como referencia durante el desarrollo de aplicaciones;
- ayuda a detectar errores frecuentes;
- proporciona un lenguaje común para hablar sobre seguridad de aplicaciones.

Por ejemplo, una organización puede utilizar el OWASP Top 10 como punto de partida para revisar una aplicación antes de ponerla en producción. Sin embargo, **cumplir con el OWASP Top 10 no significa que una aplicación sea completamente segura**. Existen numerosas vulnerabilidades y amenazas que pueden quedar fuera de esta clasificación.

Por lo tanto, debe utilizarse como una **guía de referencia y concientización**, y no como una garantía absoluta de seguridad.

6.4.3. Principales categorías del OWASP Top 10

El OWASP Top 10 agrupa los riesgos de seguridad en **categorías generales**. Cada categoría representa un conjunto de problemas que pueden afectar a las aplicaciones web. Algunas de las vulnerabilidades más importantes ya fueron estudiadas en el apartado **6.3 Principales Ataques a Aplicaciones Web**.

Las categorías de la edición **OWASP Top 10:2021** son:

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

La **autenticación** y la **gestión de sesiones** son componentes fundamentales de la seguridad de una aplicación web. Permiten verificar la identidad de los usuarios y mantener esa identidad durante las diferentes operaciones que realizan dentro de la aplicación.

Los mecanismos y factores de autenticación, como contraseñas, biometría y autenticación multifactor, fueron estudiados en las unidades anteriores. Por lo tanto, en esta unidad se analizará principalmente **su funcionamiento dentro de una aplicación web y la protección de las sesiones generadas después de la autenticación**.

Una vulnerabilidad en estos mecanismos puede permitir que un atacante acceda a una cuenta legítima o utilice una sesión perteneciente a otro usuario.

6.5.1. Autenticación de Usuarios

La **autenticación** es el proceso mediante el cual una aplicación verifica la identidad de un usuario antes de permitirle acceder a determinados recursos.

Una vez que el usuario ha sido autenticado correctamente, el servidor debe disponer de un mecanismo que permita reconocerlo en las solicitudes posteriores. Para ello se crea normalmente una **sesión**, asociada al usuario autenticado.

Es importante diferenciar:

- **Autenticación:** determina quién es el usuario.
- **Autorización:** determina qué recursos y operaciones puede utilizar ese usuario.

Por ejemplo, dos usuarios pueden estar correctamente autenticados, pero uno de ellos puede tener permisos para administrar usuarios mientras que el otro solamente puede consultar información.

6.5.2. Gestión de Sesiones

HTTP es un protocolo que, por sí mismo, no mantiene información sobre las solicitudes anteriores. Por ello, una aplicación web necesita un mecanismo que permita recordar que diferentes solicitudes pertenecen al mismo usuario.

La **gestión de sesiones** permite mantener esta asociación.

De manera simplificada, el proceso es:

1. el usuario se autentica;
2. el servidor verifica sus credenciales;
3. el servidor crea una sesión;
4. genera un identificador de sesión;
5. el identificador se envía al navegador;
6. el navegador lo utiliza en las solicitudes posteriores;
7. el servidor utiliza ese identificador para reconocer al usuario.

El identificador de sesión es un elemento **sensible**, ya que quien consiga utilizar una sesión válida podría llegar a actuar como el usuario legítimo.

Este es precisamente uno de los problemas analizados anteriormente en **6.3.4 Robo de Sesiones (Session Hijacking)**.

6.5.3. Protección de las Sesiones

La gestión de sesiones debe incorporar diferentes mecanismos de seguridad para dificultar su robo, predicción o utilización indebida. Entre las principales medidas se encuentran:

6.5.3.1. Utilización de HTTPS

La comunicación entre el navegador y el servidor debe utilizar **HTTPS**, evitando que información sensible pueda transmitirse mediante conexiones inseguras.

6.5.3.2. Identificadores de sesión seguros

Los identificadores deben ser generados de forma aleatoria y ser suficientemente difíciles de predecir. No deberían utilizarse valores simples como:

```
sesion=12345
```

ya que podrían facilitar su predicción.

6.5.3.3. Renovación del identificador

El identificador de sesión debería renovarse después de determinados eventos importantes, especialmente después de una autenticación.

6.5.3.4. Expiración de sesiones

Las sesiones deberían tener un período de validez limitado.

Por ejemplo, una aplicación puede finalizar automáticamente una sesión después de un determinado período de inactividad.

6.5.3.5. Invalidación de sesiones

Cuando el usuario selecciona **Cerrar sesión**, el identificador correspondiente debe dejar de ser válido. No basta simplemente con eliminar visualmente la página o cerrar una ventana del navegador.

6.5.3.6. Protección de las cookies de sesión

Las aplicaciones web suelen utilizar **cookies** para almacenar el identificador de sesión. Estas cookies pueden incorporar diferentes atributos de seguridad.

Atributo	Función
Secure	Hace que la cookie se envíe únicamente mediante HTTPS.
HttpOnly	Impide que determinados scripts del navegador accedan directamente a la cookie.
SameSite	Controla el envío de la cookie en solicitudes realizadas desde otros sitios.

Estos mecanismos no reemplazan otras medidas de seguridad, pero ayudan a reducir determinados riesgos relacionados con el robo de sesiones y otros ataques web.

6.5.4. Cierre y Expiración de Sesiones

Una sesión no debería permanecer activa indefinidamente. La aplicación debe establecer mecanismos para finalizarla cuando:

- el usuario selecciona **Cerrar sesión**;
- transcurre un período determinado de inactividad;
- se detecta una actividad sospechosa;
- se modifican determinados datos críticos de autenticación.

6.5.4.1. Ejemplo práctico

Supóngase que un empleado utiliza una aplicación web desde una computadora compartida. El empleado inicia sesión y posteriormente abandona el equipo sin cerrar la sesión. Si la aplicación mantiene la sesión activa indefinidamente, otra persona podría utilizar el navegador y acceder a la cuenta.

Si, en cambio, la aplicación establece un período de expiración y además permite cerrar correctamente la sesión, el riesgo disminuye considerablemente. En aplicaciones que manejan información especialmente sensible, también puede ser conveniente solicitar nuevamente la autenticación antes de realizar determinadas operaciones críticas.

6.5.5. Buenas Prácticas de Autenticación y Gestión de Sesiones

Como buenas prácticas, una aplicación web debería:

- utilizar mecanismos de autenticación seguros;
- proteger las comunicaciones mediante **HTTPS**;
- generar identificadores de sesión aleatorios y difíciles de predecir;
- renovar el identificador después de la autenticación;
- establecer tiempos de expiración adecuados;
- invalidar correctamente las sesiones cerradas;
- proteger las cookies mediante `Secure`, `HttpOnly` y `SameSite` cuando corresponda;
- solicitar autenticación adicional para operaciones especialmente sensibles;
- monitorear actividades anormales relacionadas con las sesiones;
- evitar sesiones excesivamente largas.

Estas medidas deben complementarse con los mecanismos de autenticación estudiados en las unidades anteriores.

Relación entre autenticación, sesión y autorización

La seguridad de una aplicación web requiere proteger todo este proceso. **No es suficiente con autenticar correctamente al usuario: también es necesario proteger la sesión que representa su identidad y controlar las operaciones que está autorizado a realizar.**

6.6. Protección de Aplicaciones Web

Las aplicaciones web están expuestas a Internet y, por lo tanto, pueden recibir solicitudes legítimas y maliciosas. Para reducir los riesgos es necesario incorporar diferentes mecanismos de protección. Entre los principales mecanismos relacionados directamente con la seguridad de las aplicaciones web se encuentran los **firewalls**, los **Web Application Firewall (WAF)** y el uso de **HTTPS**.

6.6.1. Firewall Tradicional

6.6.1.1. Concepto

Un **firewall** es un mecanismo de seguridad que controla el tráfico de red que entra o sale de un sistema o una red, aplicando determinadas reglas de seguridad. Su función principal es permitir o bloquear comunicaciones según criterios previamente establecidos.

Por ejemplo, una organización podría establecer reglas que permitan determinados servicios y bloqueen otros. El firewall puede utilizar diferentes características de las comunicaciones para tomar decisiones, como direcciones IP, puertos y protocolos.

En una organización, puede utilizarse para limitar el acceso desde Internet hacia los servidores internos y reducir la superficie de exposición de los sistemas.

6.6.1.2. Limitaciones frente a aplicaciones web

Un firewall tradicional es importante para proteger la infraestructura de red, pero **no está diseñado específicamente para analizar en profundidad el contenido de las solicitudes HTTP o HTTPS de una aplicación web**.

Por ejemplo, una solicitud como:

```
HTTP/1.1
GET /productos?id=15
```

puede ser considerada una comunicación válida desde el punto de vista de la red. Sin embargo, una aplicación web podría recibir dentro de una solicitud un contenido malicioso relacionado con una vulnerabilidad como **SQL Injection**.

El firewall tradicional puede permitir la comunicación porque:

- la dirección IP está permitida;
- el puerto utilizado es correcto;
- el protocolo utilizado es válido.

Pero eso no significa que la solicitud sea segura para la aplicación.

6.6.1.3. Ejemplo

Supóngase que una aplicación web funciona mediante HTTPS en el puerto correspondiente.

El firewall puede establecer:

```
Internet ⟶ Puerto HTTPS ⟶ PERMITIR
```

Esto permite que los usuarios legítimos puedan utilizar la aplicación. Sin embargo, también puede permitir que lleguen solicitudes maliciosas:

Por lo tanto, **permitir una comunicación de red no significa que el contenido de esa comunicación sea seguro**. Esta es una de las principales limitaciones de un firewall tradicional frente a las aplicaciones web.

Firewall tradicional vs. protección específica de aplicaciones

Firewall tradicional	Aplicación Web
Controla principalmente comunicaciones de red.	Requiere analizar las solicitudes dirigidas a la aplicación.
Puede controlar IP, puertos y protocolos.	Necesita comprender elementos propios de HTTP/HTTPS.
Puede bloquear determinados accesos de red.	Puede necesitar detectar patrones de ataques en las solicitudes.
Protege principalmente la infraestructura de red.	Requiere mecanismos específicos como un WAF (Web Application Firewall).

Por esta razón, el firewall tradicional **no debe considerarse un sustituto de los mecanismos de seguridad específicos para aplicaciones web**.

Un firewall puede formar parte de una arquitectura de seguridad, mientras que un **WAF** proporciona una capa adicional orientada específicamente a proteger las aplicaciones web.

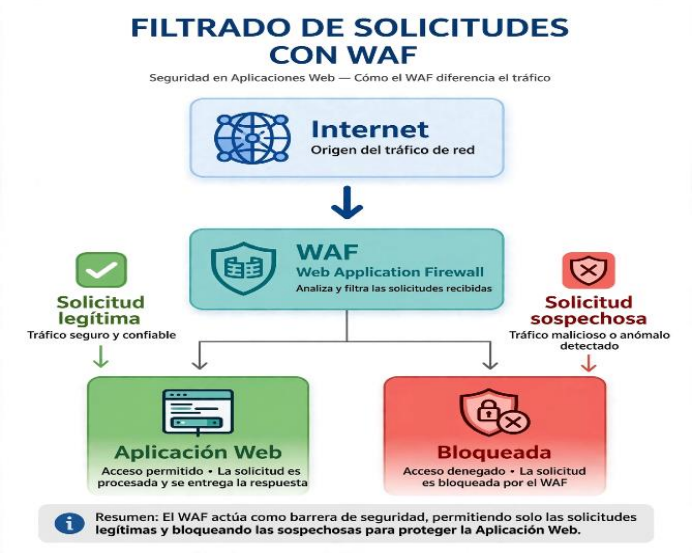
Idea fundamental: un firewall tradicional puede determinar si una comunicación de red está permitida, pero no necesariamente puede determinar si el contenido de una solicitud web es malicioso.

6.6.2. Web Application Firewall (WAF)

Un **Web Application Firewall (WAF)** es un mecanismo de seguridad diseñado específicamente para proteger las **aplicaciones web** frente a solicitudes que puedan contener comportamientos maliciosos.

A diferencia de un firewall tradicional, que se centra principalmente en controlar el tráfico de red, un WAF analiza las solicitudes dirigidas a una aplicación web y puede identificar determinados patrones asociados con ataques.

Su función puede representarse de manera simplificada:



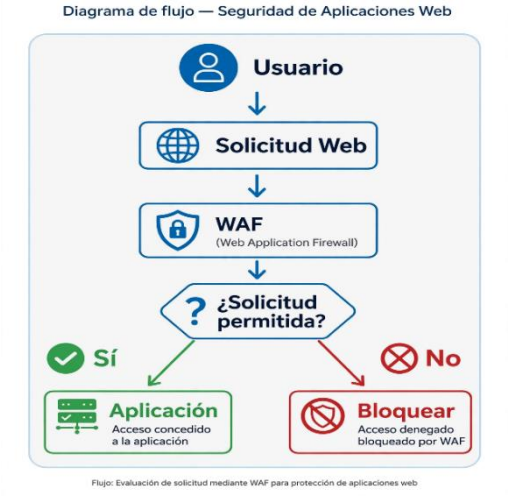
El WAF actúa, por lo tanto, como una **capa de protección situada delante de la aplicación web**.

6.6.2.1. Funcionamiento

Cuando un usuario realiza una solicitud a una aplicación protegida por un WAF, la solicitud pasa primero por este mecanismo. El WAF analiza diferentes elementos de la comunicación y los compara con reglas o patrones de seguridad.

De forma simplificada:

1. el usuario realiza una solicitud;
2. la solicitud llega al WAF;
3. el WAF analiza la solicitud;
4. determina si parece legítima o sospechosa;
5. permite o bloquea la solicitud;
6. si es permitida, llega a la aplicación.



Por ejemplo, un WAF puede utilizar reglas destinadas a detectar patrones relacionados con ataques como **SQL Injection** o **Cross Site Scripting (XSS)**, que fueron estudiados anteriormente en esta unidad.

6.6.2.2. Ejemplo práctico

Supóngase que una aplicación web permite consultar productos mediante un parámetro recibido desde el navegador.

Un usuario legítimo podría realizar una solicitud como:

```
GET /productos?id=25
```

Mientras que un atacante podría intentar enviar una solicitud manipulada con contenido asociado a una inyección. El WAF puede detectar que determinados elementos de la solicitud coinciden con patrones considerados sospechosos y bloquearla antes de que llegue a la aplicación. Esto puede reducir la posibilidad de que determinados ataques lleguen directamente a la aplicación.

6.6.2.3. Ventajas

Entre las principales ventajas de utilizar un WAF se encuentran:

- proporciona una capa adicional de protección para aplicaciones web;
- permite detectar determinados patrones de ataques;
- puede bloquear solicitudes sospechosas;
- permite establecer reglas específicas para una aplicación;
- puede generar registros de las solicitudes bloqueadas;
- puede ayudar a proteger aplicaciones frente a determinadas vulnerabilidades conocidas.

Una ventaja importante es que el WAF puede actuar como una **capa adicional de defensa**, sin necesidad de modificar inmediatamente todos los componentes de la infraestructura.

6.6.2.4. Limitaciones

El WAF también presenta limitaciones que deben tenerse en cuenta.

6.6.2.5. No reemplaza el desarrollo seguro

Una aplicación vulnerable debe ser corregida, aunque exista un WAF delante de ella.

Por ejemplo, si una aplicación presenta una vulnerabilidad de SQL Injection, la solución correcta es corregir el código, utilizando mecanismos como las **consultas parametrizadas (Prepared Statements)** estudiadas anteriormente.

El WAF puede ayudar a bloquear determinados intentos de explotación, pero **no debe considerarse la solución definitiva a la vulnerabilidad**.

6.6.2.6. Puede producir falsos positivos

Una regla demasiado restrictiva puede bloquear solicitudes legítimas. Por este motivo, las reglas del WAF deben configurarse y supervisarse correctamente.

6.6.2.7. No detecta todos los ataques

Un WAF está diseñado para determinados tipos de amenazas, pero no puede detectar todas las vulnerabilidades de una aplicación.

Por ejemplo, problemas relacionados con:

- lógica de negocio;
- permisos incorrectos;
- contraseñas débiles;
- errores de diseño;

pueden no ser solucionados simplemente mediante un WAF.

6.6.2.8. Necesita mantenimiento

Las reglas y configuraciones deben mantenerse actualizadas para adaptarse a nuevas vulnerabilidades y cambios en la aplicación.

Ventajas y limitaciones del WAF

Ventajas	Limitaciones
Añade una capa de protección	No reemplaza el desarrollo seguro
Puede bloquear solicitudes maliciosas	Puede producir falsos positivos
Detecta determinados patrones de ataque	No detecta todas las vulnerabilidades
Permite establecer reglas específicas	Requiere configuración y mantenimiento
Puede registrar solicitudes sospechosas	No soluciona problemas de lógica de negocio

6.6.2.9. WAF y Firewall tradicional

Ambos mecanismos **no son excluyentes**. Una organización puede utilizar un firewall tradicional para proteger su infraestructura de red y un WAF para proporcionar una protección específica a sus aplicaciones web.

Idea fundamental: el WAF constituye una capa adicional de seguridad orientada específicamente a las aplicaciones web, pero no sustituye la corrección de las vulnerabilidades existentes en el código o en el diseño de la aplicación.

6.6.2.10. Ejemplos de WAF

Entre las soluciones WAF existentes se encuentran **ModSecurity**, **Cloudflare WAF** y **AWS WAF**. ModSecurity es una alternativa de código abierto, mientras que Cloudflare WAF y AWS WAF son servicios administrados orientados a la protección de aplicaciones web.

6.6.3. HTTPS y Certificados Digitales

Las aplicaciones web transmiten continuamente información entre el navegador del usuario y el servidor. Esta información puede incluir credenciales, datos personales, consultas y otra información que debe protegerse durante la comunicación.

Para proteger estas comunicaciones se utiliza **HTTPS**, que permite establecer una comunicación segura entre el navegador y el servidor.

En este apartado se explicará únicamente el funcionamiento general de HTTPS, los certificados digitales, las Autoridades Certificadoras y el significado del candado que aparece en los navegadores. Los fundamentos criptográficos que hacen posible estos mecanismos serán estudiados en la **Unidad de Criptografía**.

6.6.3.1. HTTP vs. HTTPS

HTTP (Hypertext Transfer Protocol) es el protocolo utilizado para la comunicación entre clientes y servidores web. Cuando se utiliza HTTP sin mecanismos adicionales de protección, la información transmitida puede quedar expuesta durante su recorrido por la red. HTTPS agrega mecanismos de seguridad a la comunicación. De esta manera, HTTPS proporciona protección frente a determinados intentos de interceptación y modificación de la información durante la comunicación.

Comparación

HTTP	HTTPS
Comunicación web sin protección criptográfica de transporte	Comunicación web protegida
Los datos pueden quedar expuestos durante el tránsito	Los datos se protegen durante el tránsito
No permite verificar mediante certificados la identidad del servidor	Utiliza certificados digitales

HTTP	HTTPS
Menor seguridad para información sensible	Adecuado para información sensible

Por este motivo, las aplicaciones web que manejan información sensible deberían utilizar **HTTPS**.

6.6.3.2. Certificado Digital

Un **certificado digital** es un documento electrónico que permite asociar una identidad con una determinada clave utilizada en una comunicación segura.

En el contexto de HTTPS, el certificado permite que el navegador pueda comprobar que el servidor con el que está estableciendo la comunicación corresponde al sitio web indicado.

Por ejemplo, cuando un usuario accede a:

```
https://www.ejemplo.com
```

el servidor presenta un certificado digital al navegador.

El navegador verifica determinados datos del certificado, entre ellos:

- el dominio al que corresponde;
- su período de validez;
- quién lo emitió;
- si existe una relación de confianza con la entidad que lo emitió.

Si las comprobaciones son satisfactorias, el navegador puede establecer la comunicación HTTPS.

Importante: El certificado digital no significa que el sitio web sea necesariamente confiable en todos los sentidos. Principalmente permite verificar la identidad del servidor dentro del mecanismo de confianza utilizado por HTTPS.

Por ejemplo, un sitio web puede poseer un certificado válido y, aun así, contener una vulnerabilidad como SQL Injection.

6.6.3.3. Autoridad Certificadora

Una **Autoridad Certificadora (CA, Certification Authority)** es una entidad que emite certificados digitales y participa en el sistema de confianza utilizado por HTTPS.

De manera simplificada:

Las Autoridades Certificadoras permiten establecer una relación de confianza entre el sitio web y el navegador. Los navegadores y sistemas operativos disponen de información sobre determinadas Autoridades Certificadoras que consideran confiables.

6.6.3.4. El candado del navegador

Cuando un sitio utiliza HTTPS correctamente, los navegadores modernos suelen mostrar un **icono de candado** o un indicador relacionado con la conexión segura.

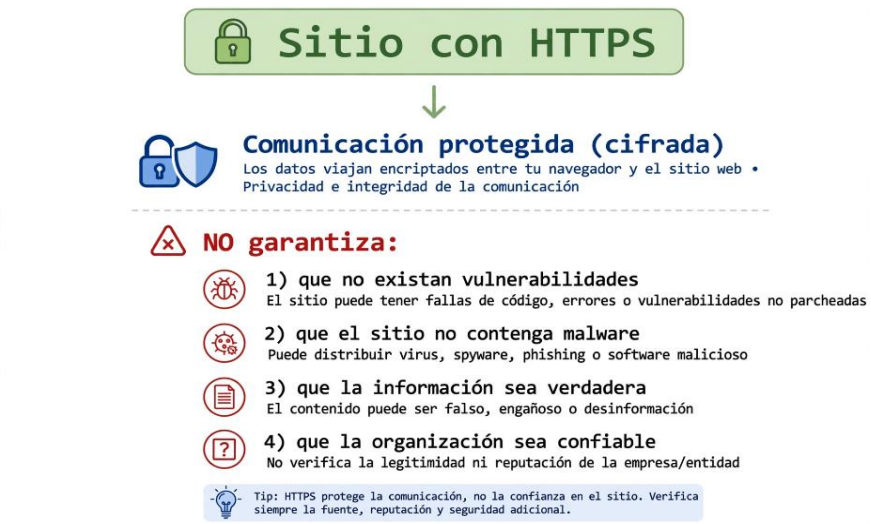
Por ejemplo:

🔒 `https://www.ejemplo.com`

El indicador permite al usuario acceder a información sobre la conexión y el certificado del sitio.

Sin embargo, el candado **no significa que el sitio sea completamente seguro o que sea legítimo en todos los aspectos**.

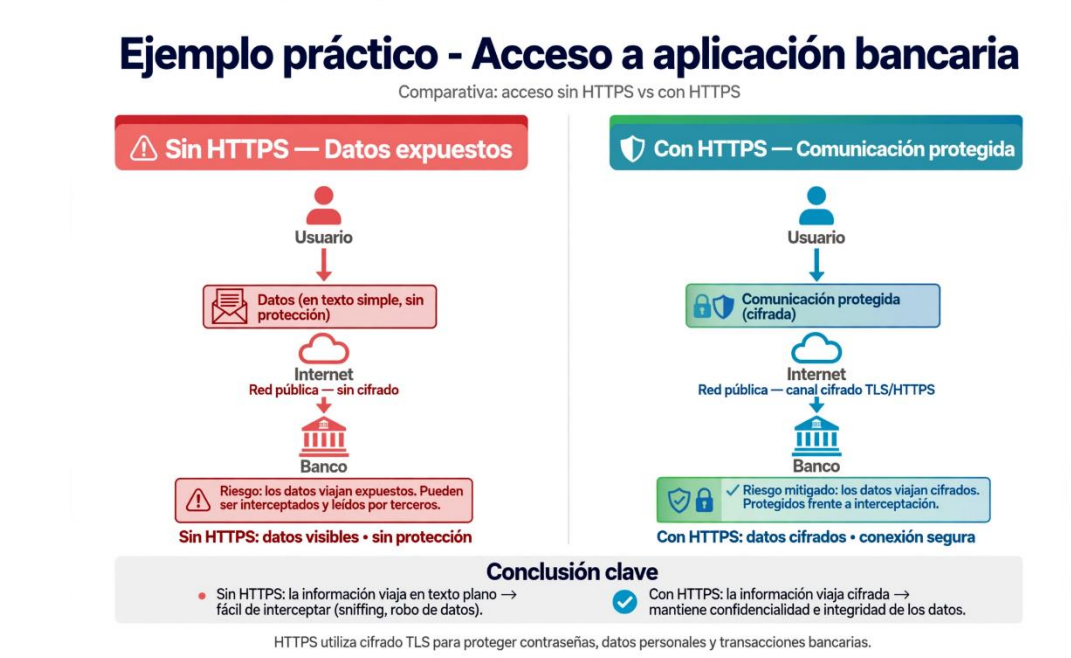
Por ejemplo:



Por lo tanto, el usuario no debe interpretar el candado como una garantía absoluta de seguridad.

6.6.3.5. Ejemplo práctico

Supóngase que un usuario ingresa a una aplicación bancaria.



El segundo escenario proporciona una protección mucho mayor para la información transmitida entre el navegador y el servidor.

Esto es especialmente importante cuando se transmiten:

- nombres de usuario;
- contraseñas;
- datos personales;
- información financiera;
- información privada.

6.6.3.6. Relación con la Seguridad Web

HTTPS es una medida fundamental para proteger las comunicaciones de una aplicación web, pero constituye solamente **una parte de la seguridad general**.

Una aplicación puede utilizar HTTPS y continuar presentando vulnerabilidades como:

- SQL Injection;
- XSS;
- problemas de control de acceso;
- robo de sesiones;
- configuraciones inseguras.

Por este motivo:

HTTPS protege la comunicación, pero no corrige las vulnerabilidades de la aplicación.

6.7. Buenas Prácticas para la Seguridad en Aplicaciones Web

La seguridad de una aplicación web requiere aplicar diferentes medidas de protección durante su diseño, desarrollo, implementación y utilización.

6.7.1. Principales buenas prácticas

Buena práctica	Objetivo
Validar las entradas	Evitar que datos manipulados por un usuario sean procesados de forma insegura.
Utilizar consultas parametrizadas	Prevenir ataques de **SQL Injection**.
Implementar controles de acceso	Evitar que un usuario acceda a recursos o funciones que no tiene autorizados.
Gestionar correctamente las sesiones	Reducir el riesgo de robo, fijación o utilización indebida de sesiones.
Utilizar HTTPS	Proteger la información transmitida entre el navegador y el servidor.
Controlar los archivos subidos	Reducir el riesgo de que se incorporen archivos maliciosos al servidor.
Controlar las solicitudes recibidas	Ayudar a reducir el impacto de determinados ataques de denegación de servicio.
Utilizar un WAF cuando corresponda	Incorporar una capa adicional de protección frente a determinados ataques dirigidos a aplicaciones web.

Estas medidas no deben considerarse independientes. Una aplicación segura requiere combinarlas de acuerdo con sus características, los datos que maneja y los riesgos a los que está expuesta.

6.7.2. Principios fundamentales

Las buenas prácticas estudiadas pueden resumirse en algunos principios fundamentales:

- **No confiar directamente en los datos proporcionados por el usuario.**

- **Validar y controlar las entradas antes de procesarlas.**
- **Separar los datos de las instrucciones**, como ocurre con las consultas parametrizadas.
- **Verificar siempre los permisos del usuario** antes de permitir una operación.
- **Proteger las sesiones** después de la autenticación.
- **Proteger las comunicaciones mediante HTTPS.**
- **Agregar capas de protección**, como un WAF, cuando las características de la aplicación lo justifiquen.

Estas medidas permiten reducir la superficie de ataque y dificultar la explotación de vulnerabilidades.