

# Detección y corrección de vulnerabilidades de aplicaciones web.



## Caso práctico



StockSnap. Portatil, programación código  
(Licencia de Pixabay)

**Julián** ha sido asignado a un nuevo proyecto de transformación digital en una importante entidad financiera. El objetivo del proyecto es poder dotar a los usuarios de la entidad de la capacidad de gestionar sus datos y realizar operaciones bancarias a través de una nueva plataforma web.

Este proyecto es muy atractivo para **Julián** y está deseando comenzar a trabajar en él, pero por otra parte hay dos cosas que le preocupan, la gran exposición que tendrá el aplicativo ya que será web y estará accesible para todo el mundo en Internet y por otra parte el hecho de que el aplicativo trabajará con varias bases de datos que tienen información personal de los usuarios.

Sabe además que hay una fundación (OWASP) y su Top10 que podrá usar para evaluar riesgos pero, aun así, es consciente de que aunque el software sea robusto y seguro necesitará ayuda y es posible que el proyecto requiera que se instalen Firewalls de aplicativo web (**WAF**) para controlar el volumen y tipo de peticiones que se reciban así como sistemas **Captchas** y demás que garanticen que sólo usuarios reales acceden al aplicativo descartando redes de **bots** o software de automatización de ataques.

Durante las últimas dos décadas estamos siendo testigos de un proceso de transformación digital a varios niveles que está provocando un gran crecimiento en la cantidad de aplicaciones de todo tipo que surgen para habilitar esta transformación. Al mismo tiempo y como consecuencia de la exposición, se está produciendo un aumento significativo de los ataques dirigidos a las aplicaciones, debido a su interacción con los datos. Todo eso está generando un canal directo para que los atacantes exploten vulnerabilidades introducidas en el desarrollo de las aplicaciones.



macblue86 para Pixabay. virus-ordenador-personal-portátil (Propia de Pixabay)



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE EDUCACIÓN, FORMACIÓN PROFESIONAL  
Y DEPORTES

[Ministerio de Educación, Formación Profesional y Deporte.](#) (Dominio público)

**Materiales formativos de FP Online propiedad del Ministerio de Educación, Formación Profesional y Deportes.**

[Aviso Legal](#)

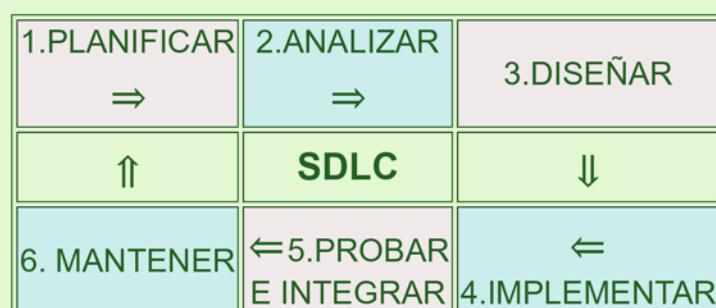


## Caso práctico

A Julián y al resto de componentes del equipo de desarrollo se les ha dado una formación inicial en ciberseguridad, que continuará durante todo el proyecto. En la entidad para la que realiza el proyecto se siguen las recomendaciones de OWASP y han empezado dando una comprensión básica de los objetivos, principios de seguridad (ver este [enlace](#)) y los mecanismos principales para implementarlos:

OBJETIVOS DE LA SEGURIDAD		
<b>CONFIDENCIALIDAD</b>  La información sólo debe estar disponible a quién debe tener acceso.	<b>INTEGRIDAD</b>  Se sabe que los datos son correctos y confiables.	<b>DISPONIBILIDAD</b>  Los usuarios con permisos pueden acceder a la información cuando la necesitan
<b>Principales mecanismos:</b> <ul style="list-style-type: none"><li>• Autenticación</li><li>• Autorización</li><li>• Gestión de sesión</li><li>• Logging</li><li>• Manejo de errores</li></ul>	<b>Principales mecanismos:</b> <ul style="list-style-type: none"><li>• Validación de datos</li><li>• Logging</li><li>• Encriptación</li></ul>	<b>Principales mecanismos:</b> <ul style="list-style-type: none"><li>• Manejo de errores</li><li>• Logging</li><li>• Encriptación</li></ul>
OTROS OBJETIVOS DE LA SEGURIDAD		
<b>Resiliencia</b>	<b>Robustez</b>	<b>Trazabilidad</b>
		<b>Autenticación</b>
		<b>Fiabilidad</b>

Además, se les ha explicado en qué consiste el **SDLC seguro (Secure SDLC en inglés)**. Julián ha entendido que casi todo el software moderno se desarrolla pasando por fases que se repiten de forma cíclica a lo largo de la vida útil de la aplicación. A continuación se muestra un ciclo de vida de desarrollo de software (**SDLC**) genérico:



Debido al creciente número de ataques, la mayoría de las empresas han adoptado un

ciclo de vida de desarrollo de software (SDLC) seguro. Julián comprende que el SDLC seguro nunca debe ser un ciclo de vida separado del SDLC, siempre debe ser el mismo ciclo de vida de desarrollo de software que antes, pero con acciones de seguridad incorporadas en cada fase.

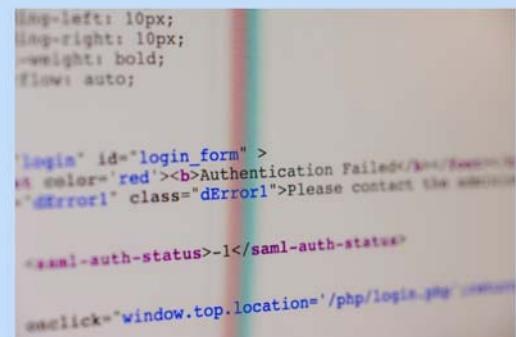
**OWASP** establece que un SDLC seguro es: "Conjunto de principios de diseño y buenas prácticas a implantar en el SDLC, para detectar, prevenir y corregir los defectos de seguridad en el desarrollo y adquisición de aplicaciones, de forma que se obtenga software de confianza y robusto frente a ataques maliciosos, que realice solo las funciones para las que fue diseñado, que esté libre de vulnerabilidades, ya sean intencionalmente diseñadas o accidentalmente insertadas durante su ciclo de vida y se asegure su integridad, disponibilidad y confidencialidad". Además, recomienda:

- ✓ Implementar un ciclo de vida de desarrollo de software seguro
- ✓ Definir claramente roles y responsabilidades.
- ✓ Proporcionar a los equipos de desarrollo la formación adecuada en seguridad de software.
- ✓ Establecer estándares de codificación seguros
- ✓ Construir una biblioteca de objetos reutilizable.
- ✓ Verificar la efectividad de los controles de seguridad (ASVS, que ya vimos en la unidad 2).
- ✓ Establecer prácticas seguras de desarrollo subcontratado, incluida la definición de requisitos de seguridad y metodologías de verificación.
- ✓ Establecer un modelo de amenazas y una matriz de amenazas (ver este [enlace](#) )

## Desarrollo seguro

El desarrollo seguro es un pilar fundamental en el mundo de la ciberseguridad, y existen diversas guías sobre como realizarlo. Nos vamos a centrar en el [OWASP Development Guide](#), que proporciona a los profesionales de la seguridad y desarrolladores una sólida base para construir aplicaciones y sistemas resistentes a las vulnerabilidades conocidas.

En la última versión estable de **OWASP Development Guide** se establecen los siguientes principios y buenas prácticas para garantizar un desarrollo seguro:



@markusspiske para Unsplash. Código ([Licencia de Unsplash](#))

## Principios a la hora de desarrollar el software:

- ✓ Minimizar los puntos de ataque.
- ✓ Establecer valores predeterminados seguros.
- ✓ Mínimo privilegio (POLP).
- ✓ Defensa en profundidad.
- ✓ Fallar de forma segura.
- ✓ No confiar en los sistemas externos: validar datos de entrada y salida.
- ✓ Segmentación.
- ✓ Evitar la seguridad por ocultación.

- ✓ Simplicidad de diseño de la seguridad.
- ✓ Corregir los problemas de seguridad correctamente.
- ✓ Registrar los eventos de seguridad

## Conjunto de buenas prácticas

Para alcanzar estos principios se recomiendan un conjunto de buenas prácticas (puedes ampliar información en los CheatSheets de OWASP, ver este [enlace](#)  ) que principalmente se agrupan en:

- ✓ Validación de entradas (y de salidas).
- ✓ Autenticación y gestión de contraseñas
- ✓ Administración de sesiones
- ✓ Autorización y control de acceso.
- ✓ Criptografía.
- ✓ Manejo de errores y logs.
- ✓ Protección de datos.
- ✓ Manejo de archivos.
- ✓ Manejo de memoria.
- ✓ Seguridad en las comunicaciones.
- ✓ Seguridad en las BBDD.
- ✓ Configuración segura de los sistemas.
- ✓ Codificación en general.



### Para saber más

Te recomiendo visitar la página de la próxima versión de la guía de desarrollo seguro para que veas el nivel de detalle que tiene; para ello, pulsa este [enlace](#) 

Además, OWASP publica multitud de documentación. Echa un vistazo a este [pdf](#)  para entender cómo proteger una aplicación web empresarial de tres niveles. En la página 14 se muestra una imagen donde se puede ver todo lo que hay que proteger: **Servidor Web, Servidor de Aplicaciones, Base de datos, los equipos, la red...**

## 2.- Entrada basada en formularios.



### Caso práctico

Particulares Empresas [← Volver](#)

Te damos la bienvenida a tu Banca Online

Documento NIF  Nº de documento

Clave de acceso

Recordar usuario [Entrar](#)

[¿Problemas con tu clave de acceso?](#)

Centro de ayuda  Oficinas y cajeros

• Alta en Banca Online • Demo  
• Seguridad

David Conde (Dominio público)

Una de las tareas que se le ha pedido realizar a **Julián** es desarrollar una nueva funcionalidad del aplicativo web de la entidad financiera, donde debe utilizar diversos **formularios**. Los formularios no son más que formas o vías de comunicación del aplicativo web que nos permiten interactuar con él proporcionándole información de entrada para que nos devuelva otro tipo de información. Suele ser la vía de comunicación estándar con un aplicativo web a nivel de usuario y se usa para muchos fines desde realizar consultas hasta poder *autenticarnos* en el aplicativo.

Julián sabe que a nivel de seguridad es uno de los principales vectores de ataque que los atacantes usan, ya que si no está debidamente configurado y protegido se puede manipular para realizar

acciones no esperadas como obtener el contenido de la base de datos, saltarnos la autenticación o incluso introducir contenido malicioso en la base de datos o en el aplicativo.

**Julián** ha estudiado las recomendaciones de OWASP para el desarrollo seguro y en especial la validación de la entrada y de la salida (ver [enlace](#) ). Además, ha podido comprobar que para la **validación en el lado del cliente o en el lado del servidor** la recomendación establece:

*Tenga en cuenta que cualquier validación de entrada de JavaScript realizada en el cliente puede ser eludida por un atacante que desactive JavaScript o utilice un proxy web. Asegúrese de que cualquier validación de entrada realizada en el cliente también se realiza en el servidor.*

Un formulario está diseñado con el propósito de que el usuario introduzca datos estructurados (nombre, apellidos, teléfono, contraseña, etc.) en las zonas del aplicativo destinadas a ese fin, para ser registradas y procesados posteriormente por el aplicativo web. Alguno de los elementos más comunes que podemos encontrar en un formulario son:

- ✓ Cajas o boxes de texto
- ✓ Listas de selección
- ✓ Checkbox o casillas de verificación
- ✓ Botones
- ✓ Desplegables informativos

Si un formulario no está configurado y protegido adecuadamente, puede ser manipulado para llevar a cabo acciones inesperadas, como obtener el contenido de la base de datos, eludir la autenticación e incluso introducir contenido malicioso en la base de datos o en la aplicación. Este tipo de ataques se llaman **ataques de inyección**, donde inyectaremos o introduciremos nuestro código malicioso para hacer que la aplicación se comporta de una manera no esperada. Por ejemplo, ese código malicioso puede estar basado en lenguajes de consulta como en los ataques de **Inyección SQL**.

Para prevenir esto es importante que el aplicativo valide correctamente las entradas de datos, que realmente compruebe si su contenido se corresponde con lo esperado (por ejemplo, sería raro que el campo "teléfono" tuviera comandos de ejecución de código), que higienice la entrada por si tuviera algún carácter extraño y también es recomendable que la interacción con la base de datos sea a través de una interfaz parametrizada o utilizar una herramienta de ORM. Para ampliar ver este [enlace](#) 



## Debes conocer

Cuando pensamos en ataques de inyección comúnmente pensamos en ataques que tienen como objetivo cambiar la manera en que el aplicativo web se comporta, un ejemplo clásico son ataques de tipo **SQL Injection**. Estos ataques se conocen como **Server Side Inyection**.

Pero dentro de la inyección hay otro tipo de ataques tanto o más críticos que no tienen como objetivo el sistema o aplicativo sino al propio usuario. Buscan robar información sensible del usuario o incluso sus *cookies* para ser re usadas más tarde de forma maliciosa. Son lo que denominamos **Client Side Inyection** y los ejemplos más claros serían los ataques de tipo **XSS** (Cross-Site-Scripting) y **HTML Inyection**.

En el primero de los tipos de ataque (Server Side Inyection), tanto los programadores como los ingenieros de seguridad dedican grandes esfuerzos en controlar la entrada de datos y su ejecución, pero en el segundo caso (Client Side Inyection), aunque el programador haya revisado todo su código y se considera seguro para el servidor puede seguir siendo un riesgo para el usuario.

Por ejemplo uno de los ataques más importantes ante el que conviene estar concienciado son los ataques de robo de sesión o *Session Hijacking* que utilizan ataques de tipo inyección en el lado del cliente para robar las cookies del usuario y poder luego usar esa cookie (que normalmente garantiza una sesión autenticada y autorizada contra un recurso) de forma maliciosa, por ejemplo para hacer transacciones de banco de forma no autorizada.



## Para saber más

Los parámetros dinámicos (también llamados bind o parámetros bind) son una alternativa para presentar los datos a la base de datos. En lugar de poner los valores directamente dentro de la sentencia SQL (o similares) y que esta se ejecute en la base de datos, se puede usar un marcador de posición específico como ?, :name o @name y proveer el valor actual de la variable. Esto provoca que el contenido introducido por el usuario solo pueda afectar de una manera limitada y controlada a la consulta o instrucción que se ejecutará en la base de datos.



## Caso práctico

Julián sabe que al publicar datos a través de un servicio web es necesario identificar quién puede acceder a dicho servicio, salvo casos específicos en que los servicios web sean de acceso público. La autenticación es el proceso de verificación del usuario o entidad, mientras que la autorización es el proceso de verificación de los permisos que tiene dicho usuario o entidad sobre un recurso específico.

Siempre se debe realizar primero la autenticación y luego la autorización. Esto debe verificarse en cada petición que realice el usuario hacia el recurso o entidad.

Por otra parte, para tener modularidad en los permisos y otorgar únicamente los permisos realmente necesarios para los usuarios o entidades. Esto se consigue mediante la asignación de **roles**.

```
background-color: #f0f0f0;
border: 1px solid #ccc;
border-radius: 5px;
padding: 10px;
width: fit-content;
margin: auto;
```

```
login" id="login_form" >
    <b>Authentication Failed</b>
    <span>Please contact the administrator</span>
    <saml-auth-status>-1</saml-auth-status>
    <a href="#" onclick="window.top.location='/php/login.php'">Click here</a>

```

@markusspiske para Unsplash. [texto código](#) (Uso gratuito bajo [Licencia Unsplash](#))

Para ambos procesos a día de hoy existen los siguientes estándares

### Autenticación Básica (Basic Auth)

La autenticación básica es el método más simple y utiliza un usuario y contraseña para identificar al usuario o entidad de origen.

Se recomienda usar este tipo de autenticación en los servicios web cuando los datos que se quiere intercambiar no tengan muchas restricciones de uso. Además, este tipo de autenticación permitirá identificar mínimamente el usuario o entidad de origen. Cabe destacar que este tipo de autenticación puede ser utilizado tanto con un tipo de servicio web de tipo REST como SOAP.

### Autenticación con certificados

La autenticación por medio de certificados se realiza solicitando el certificado digital del cliente y verificando cada mensaje enviado contra dicho certificado, lo que garantiza que el cliente del servicio web es quien dice ser. Este mecanismo es tedioso si estamos autenticando personas/navegadores, pero es especialmente útil para autenticar otro servicio web ya que se integra dentro del protocolo HTTPS y de esta forma asegura la identificación, la autenticación, la confidencialidad y la integridad. Esta modalidad se conoce como autenticación mutua TLS.

### Autenticación mediante cookie de sesión

En este modelo el usuario puede autenticarse con usuario y contraseña o por algún otro método. El servidor, a su vez, crea una sesión en su memoria o base de datos y devuelve la información del usuario a través de una cookie con el identificador de la sesión creada. En cada solicitud, se pasa el identificador de sesión y el servidor da acceso al recurso solicitado si tiene esa sesión almacenada, o no da acceso en caso contrario.

### Autenticación mediante token

En este método después de tener un usuario y contraseña validados por el servidor, se creará un token que la aplicación cliente recibirá como respuesta y que le permitirá acceder a algún recurso. A diferencia de la cookie, este token no se guarda en el servidor.

El formato de token más utilizado hoy en día es JWT (Json Web Token). Es el medio principal de autenticación en servicios de tipo REST, es un medio compacto y autocontenido para enviar datos de manera segura entre las partes en formato JSON.

## Autenticación OAuth (v1 y v2)



Chris Messina.  
[OAuth logo \(CC BY-SA\)](#)

Existen dos versiones: OAuth 1.0a y OAuth2, siendo esta última la más utilizada.

OAuth2 es un protocolo de autorización y la manera de realizar la autenticación va más allá del alcance de la especificación, por lo que se puede implementar el proceso de autenticación que se desee. Una gran mayoría de las implementaciones del estándar ya ofrecen mecanismos de autenticación.



## Para saber más

Dentro del proceso de transmisión de datos entre aplicaciones podemos optar por dos enfoques diferentes que condicionarán la manera en que nuestra aplicación se comporta y los requisitos. Estos métodos son REST y SOAP. Puedes consultar más información sobre estos métodos en el siguiente enlace de [RedHat](#).



## Autoevaluación

Identifica si las siguientes frases son verdaderas o falsas

La entrada de datos mediante formularios no es un punto crítico de las aplicaciones web.

Verdadero  Falso

**Falso**

Todo proceso de introducción de datos debe de ser revisado, securizado y monitorizado.

El riesgo de un aplicativo web viene condicionado por su exposición.

Verdadero  Falso

**Verdadero**

A mayor exposición mayor volumen de peticiones y mayor probabilidad de ser atacado por la exposición.

OWASP no evalúa el software sino que proporciona guías e identifica riesgos, definiendo también controles de seguridad.

- Verdadero  Falso

**Verdadero**

El trabajo de OWASP se centra en ayudar a entender riesgos y sus impactos.

Un cuadro de texto NO es un elemento de un formulario.

- Verdadero  Falso

**Falso**

Es uno de los principales elementos y además suelen ser blanco de ataques como XSS.

# 4.-Robo de sesión. Ataque por fuerza bruta, sniffing, propagación en URL, servidores compartidos. Métodos de prevención.



## Caso práctico

Otro problema con el que se tiene que enfrentar Julián es el **robo de sesión**. Es una técnica utilizada por ciberdelincuentes para acceder a cuentas de usuario en sistemas en línea sin permiso. Implica el secuestro de la sesión de un usuario legítimo, lo que permite al atacante asumir su identidad y realizar acciones en su nombre.

Existen diversas técnicas para realizar este robo y Julián necesita comprenderlas para poder proteger al máximo su aplicación. Los impactos del robo de sesión pueden ser graves, ya que los atacantes pueden acceder a cuentas bancarias, correos electrónicos, redes sociales y otros servicios en línea. Para prevenir el robo de sesión, es esencial implementar prácticas de seguridad sólidas (ver [enlace](#) ), como el uso de conexiones seguras (HTTPS), la autenticación de dos factores (2FA) y el monitoreo de actividad inusual en las cuentas.



[mohamed\\_hassan para Pixabay](#), [suscribir registro software](#) (Licencia [Pixabay](#))

Existen varias formas en las que se puede llevar a cabo el robo de sesión:

### Por fuerza bruta

Es un tipo de ataque en el que un atacante intenta adivinar o descifrar la contraseña de un usuario probando una amplia gama de posibles combinaciones de contraseñas hasta que encuentre la correcta y obtenga acceso a la cuenta de un usuario.

Supongamos que un atacante tiene como objetivo robar la sesión de un usuario en un servicio en línea, como una cuenta de correo electrónico. El usuario tiene una dirección de correo electrónico (por ejemplo, `usuario@example.com`) y una contraseña para acceder a su cuenta. El atacante sabe la dirección de correo electrónico del usuario y utilizando un software o un script automatizado que intenta iniciar sesión en la cuenta del usuario utilizando diferentes combinaciones de contraseñas. El atacante comienza probando contraseñas comunes, como "123456", "password", etc. Luego, el script pasa a generar contraseñas alfanuméricas más complejas o utiliza un diccionario con las palabras más utilizadas. El proceso continúa hasta que el atacante logra encontrar la contraseña correcta.

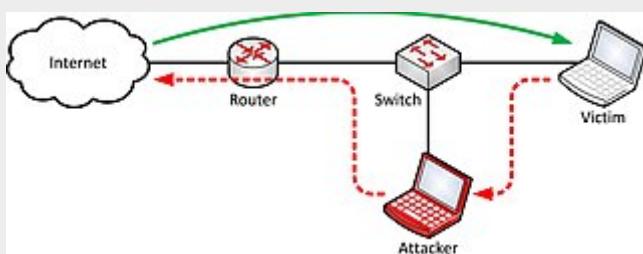


[Matt Crypto](#), [DES Cracking Machine](#) ([CC BY](#))

Este tipo de ataque puede ser lento y llevar mucho tiempo, ya que el atacante está probando

una gran cantidad de combinaciones de contraseñas. Sin embargo, si la contraseña del usuario es débil o predecible, el atacante podría tener éxito en un plazo relativamente corto. Para prevenir el robo de sesión por fuerza bruta, los usuarios deben utilizar contraseñas fuertes y únicas, que incluyan una combinación de letras mayúsculas, minúsculas, números y caracteres especiales. Además, los servicios en línea suelen implementar medidas de seguridad, como bloqueo de cuentas después de un cierto número de intentos fallidos de inicio de sesión, para proteger a los usuarios contra este tipo de ataques.

## Sniffing de sesiones



[Aliqismet\\_Aliyev. Sniffing-spoofing \(CC BY-SA\)](#)

Los atacantes pueden interceptar y capturar los datos de sesión que viajan a través de la red. Esto puede ocurrir cuando se utiliza una conexión no segura o se explotan vulnerabilidades en la red.

Supongamos que un usuario, está utilizando una red Wi-Fi pública para acceder a una aplicación web. Un atacante malicioso, también está en la misma red Wi-Fi y está ejecutando un programa de "sniffing de paquetes" que le permite interceptar el tráfico de la red. En ese tráfico de red, el atacante identifica una cookie de sesión válida que se utiliza para autenticar al usuario en su cuenta la aplicación web. Con la cookie de sesión en su poder, el atacante puede usarla para suplantar la identidad del usuario en la aplicación web, pudiendo realizar cualquier operación o incluso cambiar su contraseña si la aplicación no requiere una autenticación adicional

Para prevenir el sniffing de sesión, es fundamental utilizar conexiones seguras, como HTTPS, que cifran los datos en tránsito y dificultan la interceptación. Además, es importante evitar el uso de redes Wi-Fi públicas no seguras para acceder a aplicaciones que contienen información sensible. Los propietarios de sitios web también pueden implementar medidas de seguridad adicionales, como el uso de encabezados HTTP seguros y políticas de seguridad de contenido (CSP), para proteger a los usuarios contra ataques de sniffing de sesión.

## Cross-Site Scripting (XSS)



[christiaancole. XSS Code \(CC\)](#)

Los ataques XSS permiten a los atacantes injectar código malicioso en una página web, que se ejecuta en el navegador del usuario. Esto puede utilizarse para robar cookies de sesión u obtener información de autenticación. Este caso lo verás en la tarea.

## Phishing



[Roberto Palomo, Sabrina Rivas. Infografía: phishing \(CC BY-SA\)](#)

Los atacantes pueden engañar a los usuarios para que revelen sus credenciales de inicio de sesión a través de sitios web o correos electrónicos falsos que imitan a sitios legítimos. Supongamos que un usuario, recibe un correo electrónico aparentemente legítimo que parece provenir de su banco. El correo electrónico le informa que su cuenta bancaria ha experimentado una actividad inusual y que necesita verificar su información de inicio de sesión para proteger su cuenta. El correo electrónico contiene un enlace que supuestamente la llevará al sitio web de su banco.

El usuario, preocupado por la seguridad de su cuenta, hace clic en el enlace y llega a una página web que se parece exactamente al sitio web de su banco. En la página falsa, se le pide que inicie sesión con su nombre de usuario y contraseña, lo que María hace creyendo que está en el sitio legítimo de su banco.

Una vez que el usuario ingresa sus credenciales, el atacante detrás del sitio de phishing captura su información de inicio de sesión. Después de esto, redirige al usuario a la página de inicio de sesión real del banco, lo que puede hacer que ella crea que ha resuelto el problema de seguridad. El atacante ahora tiene acceso a las credenciales del usuario y puede utilizarlas para acceder a su cuenta bancaria.

Para evitar caer en un ataque de phishing de sesión, es esencial que los usuarios sean cautelosos y verifiquen la autenticidad de los enlaces y sitios web que visitan. Siempre es recomendable escribir la dirección del sitio web directamente en el navegador en lugar de hacer clic en enlaces en correos electrónicos no solicitados. Además, los bancos y otros servicios en línea suelen proporcionar consejos de seguridad, como no divulgar nunca información confidencial por correo electrónico.

## Propagación por URL

Incluye una variedad de técnicas o conceptos relacionados con la propagación de URLs maliciosas o engañosas en ataques. Aquí hay algunos conceptos relacionados:

- ✓ **Phishing por URL:** El phishing por URL se refiere a la práctica de enviar vínculos a sitios web o páginas falsas que imitan sitios web legítimos con el fin de engañar a los usuarios para que revelen información confidencial, como contraseñas o datos financieros. Los atacantes pueden propagar estas URL maliciosas a través de correos electrónicos, mensajes de texto u otras formas de comunicación.
- ✓ **Difusión de malware por URL:** Los atacantes pueden propagar URLs maliciosas que conducen a sitios web o descargas de archivos que contienen malware. Los usuarios que hacen clic en estas URLs pueden verse afectados por virus, troyanos o ransomware.
- ✓ **URL acortadas:** Los servicios de acortamiento de URL, como Bitly o TinyURL, se utilizan para convertir URLs largas en versiones más cortas y fáciles de compartir. Los atacantes pueden aprovecharse de esto para ocultar las URL maliciosas y hacer que parezcan seguras.
- ✓ **Propagación de enlaces maliciosos en redes sociales:** Los atacantes a menudo utilizan plataformas de redes sociales para compartir enlaces maliciosos o engañosos que redirigen a los usuarios a sitios web o contenido peligroso.



## Caso práctico

Entendemos por vulnerabilidad un fallo o debilidad en el código que permite ser explotado para poder realizar acciones distintas al fin del aplicativo, en este caso de forma maliciosa.

Como vimos anteriormente la fundación OWASP identifica los diez riesgos principales, que podemos relacionar con ataques y vulnerabilidades. Vamos a analizar en detalle el TOP-5

- 1.- Pérdida del control de acceso (Broken Access Control)
- 2.- Fallos criptográficos (Cryptographic Failures)
- 3.- Inyección (Injection)
- 4.- Diseño inseguro (Insecure Design)
- 5.- Configuración de seguridad defectuosa (Security Misconfiguration)

Si bien OWASP en su lista incluye más, este Top5 engloba la mayoría de los ataques que se producen diariamente en los aplicativos web, y Julián sabe que si consigue proteger su aplicativo con las suficientes contramedidas tanto a nivel de software seguro como de tecnología adicional podrá dotarlo de gran robustez.



@lucabravo para unsplash. [computadora-portatil-gris-encendida](#) (Uso gratuito Licencia Unsplash)

Veamos cada una de las vulnerabilidades:

### Perdida del Control de Acceso

Para ver en profundidad pulsa este [enlace](#)

**Descripción:** El control de acceso hace cumplir la política de modo que los usuarios no pueden actuar fuera de sus permisos previstos.

**Consecuencias de la vulnerabilidad:**

- ✓ Divulgación de información no autorizada
- ✓ Modificación o la destrucción de todos los datos.
- ✓ Realización de una función fuera de los límites del usuario.

### Fallos criptográficos

Para ver en profundidad pulsa este [enlace](#)

**Descripción:** la información sensible, debe estar almacenada o debe ser trasmitida de forma que no sea expuesta indebidamente.

**Consecuencias de la vulnerabilidad:** exponer información sensible, que en muchas ocasiones conlleva consecuencias penales para los encargados de asegurar la protección de esa información y perdida de reputación de la empresa.

## Inyección

Para ver en profundidad pulsa este [enlace](#)

**Descripción:** Inicialmente comprendía sólo la inyección SQL pero se ha ampliado a cualquier tipo de inyección. Las CWE ((Common Weakness Enumeration o Enumeración Común de Debilidades, en español) incluidas son CWE-79: Secuencia de Comandos en Sitios Cruzados (XSS), CWE-89: Inyección SQL, y la CWE-73:Control Externo de Nombre de archivos o ruta.

**Recomendación:** se debe realizar una validación y limpieza de toda la entrada y salida de la aplicación.

## Diseño inseguro

Para ver en profundidad pulsa este [enlace](#)

**Descripción:** El diseño inseguro es una categoría amplia que representa diferentes debilidades, expresadas como "diseño de control faltante o ineficaz". Las principales CWE incluidas son CWE-209: Generación de mensaje de error que contiene información confidencial, CWE-256: Almacenamiento desprotegido de credenciales, CWE-501: Violación de las fronteras de confianza y CWE-522: Credenciales protegidas insuficientemente.

## Configuración de seguridad defectuosa

Para ver en profundidad pulsa este [enlace](#)

**Descripción:** Sin un proceso concertado y repetible de configuración de la seguridad de las aplicaciones, los sistemas corren un mayor riesgo.

### Ejemplos de vulnerabilidades incluidas:

- ✓ Falta un “hardening” de seguridad apropiado en cualquier parte de la pila de aplicaciones o se configuran incorrectamente los permisos en los servicios en la nube.
- ✓ Se han habilitado o instalado funciones innecesarias (por ejemplo, puertos, servicios,

páginas, cuentas o privilegios innecesarios).

✓ Las cuentas por defecto y sus contraseñas siguen habilitadas y sin cambios.

✓ El manejo de errores revela rastros de pila u otros mensajes de error demasiado informativos para los usuarios



## Debes conocer

El **Instituto Nacional de Ciber Seguridad** (INCIBE) dispone de un artículo muy interesante sobre el Top10 de OWASP. Puedes consultararlo [aquí](#). 

# 6.- Almacenamiento seguro de contraseñas.



## Caso práctico

Los aplicativos web deberían de trabajar y sobre todo almacenar los menores datos posibles, es decir mediante el principio del menor acceso posible. Pero a veces por diversas cuestiones necesitan almacenar contraseñas o datos sensibles.

La mejor manera de protegerlos es mediante funciones de cifrado y funciones de hash. También los protocolos usados (incluidos los de autenticación y autorización) usan sus propios mecanismos de protección, así como sistemas de cifrado y funciones hash.

En la aplicación que está desarrollando **Julián** es necesario almacenar datos sensibles y contraseñas así que ha tenido que investigar sobre el tema, principalmente las recomendaciones de OWASP.



[mohamed\\_hassan para Pixabay. suscribir registro software](#) (Licencia [Pixabay](#))

### Cifrado

Permite cifrar la información mediante sistemas de cifrado de clave simétrica o asimétrica. Este último es el más seguro pero requiere de más carga computacional.

### Funciones Hash

Permite en vez de guardar las credenciales en texto plano guardar una imagen *hash* de la contraseña. Este tipo de funciones hace que pequeños cambios en la contraseña se reflejen en grandes cambios en su hash, resultando por tanto muy complicado hacer el análisis criptográfico y obtener la contraseña del usuario. Hoy en día los algoritmos hash mas usados son SHA1, SHA2, SHA3. Antiguamente se usaba el algoritmo MD5 pero actualmente no se considera seguro por su grado de colisiones. Es importante remarcar que las funciones hash hoy en día se usan con *salt* que no es mas que bits aleatorios que se usan para generar imágenes hash aún mas aleatorias y evitar su cripto análisis.

### Protocolos

Muchos de los protocolos que se usan hoy en día en programación y comunicación de aplicativos ya llevan embebidas funciones hash y sistemas de cifrado. Por ejemplo openssl gestiona la comunicación segura de las comunicaciones cliente-servidor.

### RECOMENDACIÓN

La mejor recomendación de seguridad no es el tipo de sistema de cifrado o función hash a usar sino evitar almacenar o gestionar cualquier tipo de información sensible a menos de que sea estrictamente necesario. OWASP establece unas recomendaciones en el caso de tener que almacenar contraseñas, que puedes consultar en este [enlace](#).



tumisu en Pixabay. [ssl-https-la-seguridad-ordenadores](https://ssl-https-la-seguridad-ordenadores)  
(Uso gratuito [Licencia Pixabay](#))



## Reflexiona

Si te has quedado con ganas de más y te surgen dudas sobre que funciones hash son preferibles y cuáles son las razones, seguro que encuentras interesante este [enlace](#) al debate planteado sobre el uso de **MD5**. ¿Qué ventajas observas en MD5? ¿Y qué vulnerabilidades? ¿Cuáles son las alternativas que se proponen? ¿Y para qué usos?



## Caso práctico

Si bien proteger nuestro software mediante una revisión exhaustiva que evite vulnerabilidades y un diseño seguro es una de las mejores recomendaciones a veces no es suficiente. Muchas veces no podemos contemplar todas las variables y aunque lo hagamos podemos sufrir ataques de programas maliciosos que son capaces de generar miles de peticiones e intentos de explotar nuestro software por segundo que hace que sea complicado de prevenir.



StockSnap. Portatil\_programación código  
(Licencia de Pixabay)

Julián quiere proteger su desarrollo lo máximo posible, así que ha investigado medidas adicionales de seguridad como **HSTS**, **CSP** o el uso de **Captchas**.

Otras medidas adicionales de seguridad son:

### CAPTCHA

Son las siglas de Completely Automated Public Turing test to tell Computers and Humans Apart. Son pruebas de tipo desafío-respuesta que permite a los aplicativos web saber cuándo hay un usuario real de una máquina o software. Este tipo de control permite reducir el uso de ataques de tipo **botnet** o software automatizado contra nuestro aplicativo. Te recomiendo ver este [enlace](#)



Google. [Logo Google Recaptcha](#)  
(Dominio público)

### HSTS (*HTTP Strict Transport Security* o *HTTP con Seguridad de Transporte Estricta*)

Son una serie de mecanismos que soportan los principales servidores y navegadores web para evitar el robo de sesiones y datos confidenciales de los usuarios. Surgió en su momento para evitar que un atacante interceptaría una comunicación entre cliente-servidor y robase o espiese su contenido o degradase la seguridad. Te recomiendo visitar este [enlace](#) .

### CSP (Content Security Policy)

Es un estándar que define un conjunto de orígenes pre-aprobados desde los que un navegador puede cargar recursos cuando visita un sitio web. También indica desde dónde podemos ejecutar scripts, imágenes y demás. Un navegador compatible con CSP solo ejecutará scripts de los archivos fuentes especificados en esa lista blanca de dominios, ignorando completamente cualquier otro script (incluyendo los **scripts inline** y los atributos de HTML de manejo de eventos). Como medida extrema de protección, los sitios que nunca requieran ejecutar scripts, pueden optar por rechazar globalmente la ejecución de scripts.

Surgió y es efectivo para dar respuesta a la gran cantidad de ataques de tipo Cross Site Scripting (XSS). Requiere la modificación del encabezado para prevenir la ejecución o redirección maliciosa del XSS. Te recomiendo visitar este [enlace](#) .



## Autoevaluación

El software seguro es suficiente para que nuestro aplicativo web esté protegido.

- Verdadero  Falso

**Falso**

Un software seguro no es suficiente, necesitamos controles y mecanismos adicionales tanto a nivel de software como de tecnología.

Un captcha ayuda a prevenir software malicioso atacando nuestro aplicativo web.

- Verdadero  Falso

**Verdadero**

Un captcha nos ayuda a separar el tráfico legitimo de usuarios de actividad automática maliciosa.

CSP ayuda a evitar los ataques de tipo XSS del lado cliente.

- Verdadero  Falso

**Verdadero**

Aporta una capa más de protección provocando que no se pueda ejecutar código aunque exista la vulnerabilidad.

HSTS ayuda a prevenir los ataques de tipo Man-In-The-Middle

- Verdadero  Falso

**Verdadero**

El principal cometido de HSTS es impedir que un atacante convierta una conexión HTTPS en HTTP



## Caso práctico

Si bien desde el punto de vista del programador tenemos el objetivo de garantizar que el código es seguro y se comporta de manera controlada y esperada, es posible que con todo esto no podamos aun así estar seguros, ya que muchas veces intervienen otros componentes como bases de datos que se escapan del control del programador o el aplicativo recibe miles de peticiones por segundos que hace muy difícil garantizar su comportamiento esperado.

Julián ha oido hablar de los cortafuegos de aplicación Web, o **WAF**. Ha decidido investigar en profundidad porque inicialmente pensaba que son como un cortafuego tradicional con algún tipo de capacidad adicional, pero nada más lejos de la realidad. Los WAFs son dispositivos complejos, que requieren de unos conocimientos elevados para administrar y que proporcionan mucha capacidad de detección y bloqueo de ataques.



[stocksnap para Pixabay. portátil, programación, código](#)  
(Propia de Pixabay)

Un **WAF** desempeña un papel crucial en la defensa de los servidores de aplicaciones web al **operar en el backend**. Su principal objetivo radica en salvaguardar la seguridad del servidor web al llevar a cabo análisis, filtrado y bloqueo de paquetes de solicitud HTTP/HTTPS. El WAF examina cada solicitud dirigida al servidor antes de que alcance la aplicación, asegurándose de que cumpla con las reglas predefinidas del firewall.

La diferencia principal de los WAFs con un cortafuego tradicional es que los WAFs saben cómo funciona un aplicativo web, cómo interacciona con la base de datos y qué tipo de datos suelen entrar y salir del aplicativo, detectando ataques de manera más sencilla. ¿Cómo lo consiguen? De una manera sencilla: cuando instalamos un WAF se dedica a estar durante meses aprendiendo cómo funciona la aplicación, qué tipo de datos maneja, cuándo se conectan los usuarios, qué datos devuelve y demás. Esto provoca que puedan detectar cualquier tipo de anomalía, ataque o parámetro malicioso, bloqueando el ataque.

Además, son capaces de detectar cuando no se trata de usuarios reales sino redes de **bots**, bloqueando estas conexiones y reduciendo el impacto de los ataques de denegación de servicio.

Las características WAF pueden ser implementadas:

- ✓ Como software mediante programa independiente, o como complemento de un servidor web (por ejemplo, el módulo ModSecurity en los servidores Apache o Nginx) o como "plugin" de un gestor de contenidos (por ejemplo, WP WAF en Wordpress).
- ✓ Como un servicio de nube: todos los "cloud" disponen de servicio WAF (por ejemplo, el AWS WAF ).
- ✓ Con un hardware específico (por ejemplo, los Barracuda web application firewall 860)

Al final disponer de un WAF no es más que otra medida más que ayuda a mejorar la seguridad de nuestro aplicativo, de forma externa al software en este caso.



## Para saber más

Hay muchas maneras de desplegar un WAF y muchos modos de funcionamiento, por ejemplo, puede estar únicamente detectando ataques o puede estar detectando y bloqueando. Por otra parte tenemos WAFs físicos en forma de *appliance* o máquina física y modelos en nube que podremos desplegar en cuestión de minutos mediante algunos cambios (principalmente en los registros DNS y en algunos certificados)

En el siguiente vídeo puedes ver las principales características de un WAF.

[https://www.youtube.com/embed/qCIFZRlbQu0?  
si=5upNuINMDZqCCNH3](https://www.youtube.com/embed/qCIFZRlbQu0?si=5upNuINMDZqCCNH3)

[@AlbertoLopez. ¿Qué es WAF o un Firewall de Aplicaciones Web? ¿Cómo funciona WAF y Para Qué Sirve?](#) ([Licencia estándar de YouTube](#))

