

Puesta en Producción Segura

Unidad 3.

Autenticación y Gestión de Sesiones



JUNTA DE EXTREMADURA

★ **R** Plan de
Recuperación,
Transformación
y Resiliencia



Financiado por
la Unión Europea
NextGenerationEU

MINISTERIO
DE EDUCACIÓN,
FORMACIÓN PROFESIONAL
Y DEPORTES

"Los atacantes no rompen el cifrado, atacan donde las personas han dejado la puerta entreabierta."

— Adaptación inspirada en Ross Anderson

Objetivos

-



Contenidos

01 Autenticación y gestión de Sesiones

02 Métodos de autenticación

03 Almacenamiento y transmisión de contraseñas

04 Vulnerabilidades en autenticación y autorización.

05 Vulnerabilidades en la gestión de sesión.

06



01

Autenticación y gestión de Sesiones



Autenticación y gestión de sesiones

¿Qué es la autenticación?

- Proceso de verificar la identidad de un usuario en un sistema.
- Métodos comunes: Contraseñas, biometría, autenticación multifactor (MFA).
- Otro factor a tener en cuenta en la autenticación es la experiencia del usuario, que suele ser inversamente proporcional a la seguridad.

Autenticación y gestión de sesiones

¿Qué es la gestión de sesiones?

- Mecanismo para mantener el estado de autenticación de un usuario después del inicio de sesión.
- Depende de cookies, tokens y mecanismos de almacenamiento seguro en el servidor.

Problemas comunes en autenticación y sesiones

Las vulnerabilidades en los mecanismos de autenticación permiten a un atacante obtener las credenciales de los usuarios de la aplicación. Este tipo de ataques explotan debilidades como las siguientes:

- La aplicación permite especificar contraseñas **poco seguras**.
- No se **limita el número de intentos** de autenticación fallidos.
- Las contraseñas se **almacenan de forma poco segura** o se **transmiten en claro** a través de medios poco seguros por lo que son filtradas después de un ataque.
- **Secuestro de sesión** mediante ataques como CSRF, MiTM o XSS.
- **Fijación de sesión (Session Fixation)** reutilizando identificadores de sesión.
- Uso de cookies inseguras (sin atributos **HttpOnly, Secure o SameSite**).

Mensajes de error de autenticación

También es una buena práctica no desvelar qué parte de la autenticación no es correcta.

Por lo tanto, es recomendable NO mostrar un mensaje de error diferente cuando el usuario no existe y cuando la **contraseña no es la correcta**. Con la información de que el usuario no existe, el atacante puede averiguar más fácilmente nombres de usuario válidos.



02

Métodos de autenticación

Tipos de autenticación

Los principales tipos de autenticación se basan en:

1. Algo que sabes (**Knowledge-based**).
2. Algo que tienes (**Possession-based**).
3. Algo que eres (**Biométrico / Inherence-based**).
4. Algo que haces (**Behavioral-based**).
5. Algun lugar donde estás (**Location-based**).



Fuente de la imagen:
<https://easy-peasy.ai/ai-image-generator/images/password-security-safe-computing>

Autenticación Knowledge-based

Se basan en algo que el usuario conoce:

- Contraseñas.
- PINs.
- Patrones de desbloqueo.
- Preguntas de seguridad (no recomendadas hoy en día).

Autenticación Possession-based

Se basan en algo que el usuario posee:

- Tokens físicos (ej. YubiKey, RSA SecurID)
- Tarjetas inteligentes (smartcards)
- Aplicaciones móviles de OTP (Google Authenticator, Authy, etc.)
- SMS/Email (menos seguro, pero aún usado).

Autenticación Inherence-based

Se basan en características de la anatomía del usuario:

- Huella dactilar.
- Reconocimiento facial.
- Reconocimiento de voz.
- Escáner de iris.



Autenticación Behavioral-based

Autenticación Behavioral-based

- Patrones de tecleo (keystroke dynamics).
- Forma de andar (gait recognition).
- Uso del ratón o gestos en pantalla.

Autenticación Location-based

- Geolocalización (ej. acceso solo desde una red corporativa o país)
- Dirección IP o red conocida



Métodos de autenticación

También podemos encontrar la clasificación según el número de métodos utilizados:

- Single-Factor Authentication (SFA): uso de un único método (ej. solo contraseña).
- Multi-Factor Authentication (MFA): combinación de dos o más factores distintos.
- Passwordless Authentication: autenticación sin contraseñas, usando enlaces mágicos, biometría o llaves de seguridad.

Contraseñas

- El usuario introduce un identificador (usuario/email) y una contraseña que se compara con la almacenada por la aplicación de forma segura.
- Si un usuario especifica una contraseña poco segura, es posible que un ataque de fuerza bruta basado en un diccionario logre averiguarla.
- Para evitarlo, es recomendable que la política de contraseñas contenga, al menos, las siguientes reglas:
 - **Longitud mínima** de la contraseña.
 - Aumentar la **complejidad de la contraseña** combinando distintos tipos de caracteres: Letras (mayúsculas y minúsculas), números y otros símbolos.
 - Rotación y limitaciones de reutilización.

Ventajas y desventajas de las contraseñas

Ventajas:

- Simples de implementar.
- Universales.

Desventajas:

- Pueden ser débiles (fáciles de adivinar o filtrar).
- Requieren políticas de seguridad (longitud, complejidad, rotación).
- Suelen reutilizarse en múltiples servicios.



Contraseñas

- Adicionalmente, podemos validar que la contraseña no esté incluida dentro de alguna lista conocida de contraseñas poco seguras, por ejemplo, "Top 10000 worst passwords":
<https://github.com/danielmiessler/SecLists/tree/master/Passwords>
- La siguiente guía del NIST, contiene una lista más detallada de reglas para establecer las políticas de:
 - Contraseñas robustas.
 - Rotación de contraseñas.

<https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret>

Contrasen as de un s o uso (OTP)

- Una contrase a de un solo uso u OTP (del ingl s One-Time Password) es una contrase a v lida solo para una autenticaci n. La OTP soluciona una serie de deficiencias que se asocian con la tradicional (est tica) contrase a.
- En contra de las contrase as est ticas no son vulnerables a ataques de REPLAY y m s resistentes frente a ataques de fuerza bruta.
- Por regla com n se env an a correo el ctrónico, dispositivo m vil, app, etc.

Autenticación Sign-On (SSO)

- La autenticación Sign-On o SSO es un sistema que permite a un usuario autenticarse una sola vez y acceder a múltiples aplicaciones o servicios sin volver a introducir credenciales.
- El usuario inicia sesión en un proveedor central (ej. Active Directory, Google Workspace, Azure AD, Google). Luego, al acceder a otros servicios, estos confían en esa sesión central.
- Un ejemplo son los servicios de Google. Una vez autenticados podemos acceder a Google Docs, Gmail, Google Drive, etc.

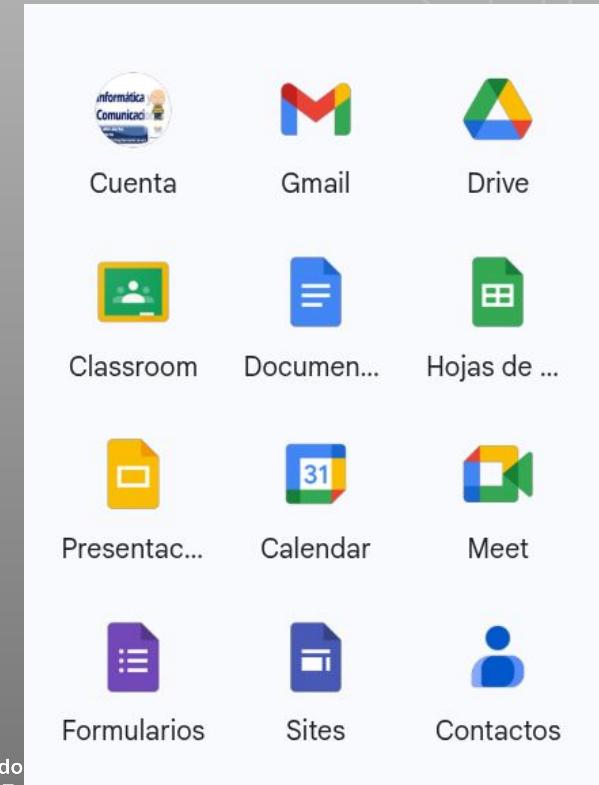
Ventajas y desventajas de SSO

Ventajas:

- Mejor experiencia de usuario (menos inicios de sesión).
- Facilita la gestión de accesos.
- Menos contraseñas que recordar → menos reutilización.

Desventajas:

- Si el proveedor SSO se compromete, todos los servicios quedan en riesgo.
- Complejidad técnica en la implementación.



Autorización OAuth 2.0

- OAuth es un **protocolo de autorización**, no de autenticación.
- Permite que una aplicación acceda a recursos de otra sin que el usuario comparta sus credenciales.
- El usuario da consentimiento para que un servicio (cliente) acceda a sus datos en otro servicio (proveedor).
- Se usan **tokens de acceso** en lugar de contraseñas.
- Un ejemplo la autenticación usando cuenta Google de diferentes webs.



The screenshot shows a login form for 'Informática y Comunicaciones' at 'informatica.iesvalledeljerteplasencia.es'. A red box highlights the error message: 'Su sesión ha excedido el tiempo límite. Por favor, acceda de nuevo.' Below the message are input fields for 'Nombre de usuario o correo electrónico' and 'Contraseña', and a blue 'Acceder' button. To the right, there's a link '¿Olvidó su contraseña?'. At the bottom, there's a section titled 'Identifíquese usando su cuenta en:' with a 'Acceder con cuenta Google' button.

Ventajas y desventajas de OAuth 2.0

Ventajas:

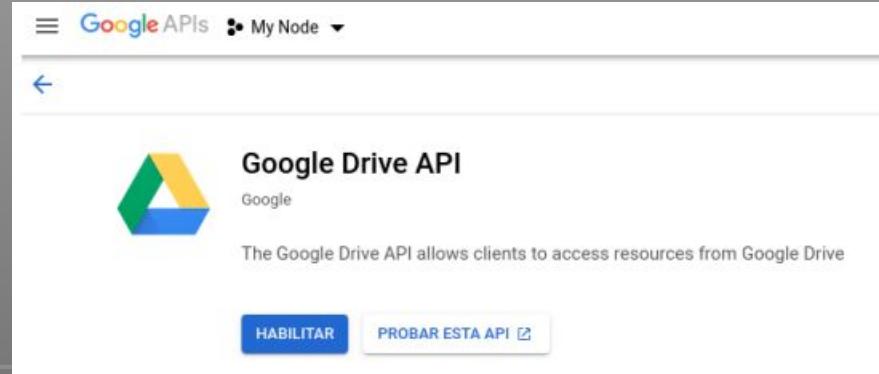
- El usuario no comparte su contraseña con terceros.
- Permite permisos granulares (ej. "solo leer tus contactos").
- Base de sistemas de autenticación moderna (ej. OpenID Connect para identidad).

Desventajas:

- Puede ser complejo de implementar correctamente.
- Malas configuraciones → vulnerabilidades (ej. OAuth abuse).

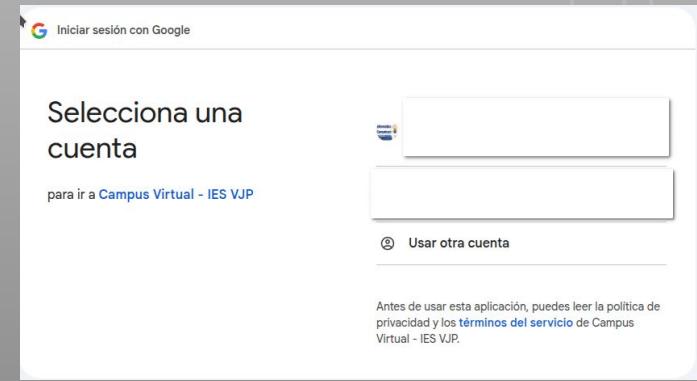
Implementación OAuth 2.0

- Para utilizar autenticación OAuth 2.0 en nuestras aplicaciones web tenemos que utilizar una API de Google o el proveedor de identidad que elijamos, que es la que se conectará al proveedor y nos pasará un token con la autorización.



OpenID Connect (OIDC)

- **OpenID Connect (OIDC)** es un protocolo que añade una capa de identidad sobre OAuth 2.0.
- Permite autenticar usuarios (probar quién eres) usando un proveedor de identidad.
- OAuth 2.0 está enfocado a la autorización, mientras que OpenID hacia la autenticación.
- Un ejemplo es el inicio de sesión en una web con “Login de Google/Microsoft/Github”.



OAuth 2.0 vs OpenId

“OAuth responde: ¿qué puede hacer esta app en mi nombre?
OpenID Connect responde: ¿quién eres tú?”

FIDO (Fast IDentity Online)

- La autenticación FIDO (Fast IDentity Online) es un conjunto de estándares para una autenticación rápida, simple y sólida.
- Cuando un usuario se registra en un servicio «online» que emplea el estándar FIDO, el sistema genera una pareja de claves criptográficas, de forma que la clave privada se conserva en el «hardware» del dispositivo y la clave pública se guarda en el servicio «online».
- Autenticación se realiza con operación matemática con clave privada, una vez que cliente ha desbloqueado el dispositivo.



JSON web tokens (JWT)

- JSON Web Token (abreviado JWT) es un estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o *claims* en inglés.
- Por ejemplo, un servidor podría generar un token indicando que el usuario tiene privilegios de administrador y proporcionarlo a un cliente.
- El cliente entonces podría utilizar el token para probar que está actuando como un administrador en el cliente o en otro sistema.
- El token está firmado por la clave del servidor, así que el cliente y el servidor son ambos capaz de verificar que el token es legítimo

Ejemplo de JWT

- Un JWT se compone de tres partes:
xxxxx.yyyyy.zzzzz
 - **Header** (encabezado): tipo de token + algoritmo de firma.
 - **Payload** (carga útil): los datos (claims).
 - **Signature** (firma): asegura la integridad.
- La decodificación del siguiente token se muestra a la derecha:

"eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.
eyJzdWliOilxMjM0NTYiLCJuYW1lIjoiJuan
PérezliwiaWF0ljoxNjE2MjM5MDlyfQ.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7
HgQ"

Header (Base64URL → JSON):

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Payload (Base64URL → JSON):

```
{  
  "sub": "123456",  
  "name": "Juan Pérez",  
  "iat": 1616239022  
}
```

Signature:
HMACSHA256(

base64UrlEncode(header) + "." +
base64UrlEncode(payload),
Clave_secreta

)

Autenticación passwordless

- La tendencia actual lleva a la eliminación de contraseñas:
- FIDO2.
- Biometría
- Magic Link (enlace mágico por email).



Magic Link

Funcionamiento:

- El usuario introduce solo su correo electrónico en la aplicación o web.
- El sistema genera un token de un solo uso y lo envía al email del usuario como un enlace.
- Ejemplo: <https://app.com/login?token=XYZ123>
- El usuario abre su correo, hace clic en el enlace, y automáticamente queda autenticado en la aplicación.

Ventajas: Sencillo, sin contraseña que recordar, reduce riesgo de phishing.

Inconvenientes: Seguridad depende de email, el enlace debe de ser temporal y de un sólo uso, es mejor que utilizar contraseñas débiles.



03

Almacenamiento y transmisión de contraseñas



Hashing vs. Encriptación

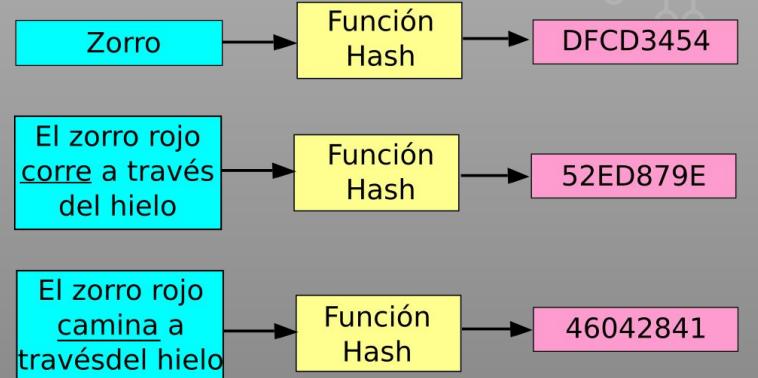
- Proceso autenticación:
 - Usuario introduce contraseña.
 - Aplicación comprueba si contraseña coincide con la guardada en BBDD.
- Si contraseñas se guardan o transmiten sin protección (en claro) y atacantes acceden, la información queda comprometida.
- **Cifrado** es una función bidireccional.
- **Hashing** es una función unidireccional (es decir, no es posible «desencriptar» un hash y obtener el valor original),



Hash

- Para no almacenar las contraseñas en claro, una técnica muy utilizada es la de hash y sal (Hash and Salt)
- Una función criptográfica de hash es un algoritmo matemático que transforma un bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija.

Entrada



Valor Hash

Zorro → Función Hash → DFCD3454

El zorro rojo corre a través del hielo → Función Hash → 52ED879E

El zorro rojo camina a través del hielo → Función Hash → 46042841

Fuente de la imagen:
https://es.wikipedia.org/wiki/Funci%C3%B3n_B3n_hash#/media/Archivo:Hash_function2-es.svg

Hash

- Las funciones de hash son **unidireccionales** (no reversibles), por lo tanto, lo que tratan de garantizar es que con los datos de salida tras aplicar la función, no se puedan obtener los datos de entrada.
- Las funciones de hash se utilizan, entre otros, para:
 - Asegurar que un fichero no se ha modificado en una transmisión
 - Firmar digitalmente un documento
 - Hacer ilegible una contraseña



¿Es mejor almacenar un hash o la contraseña cifrada?

- Los algoritmos de cifrado (como por ejemplo los algoritmos de clave simétrica) son reversibles. Esto significa que si el atacante obtiene la clave de cifrado junto con la base de datos de usuarios, podría obtener las contraseñas.
- Como los hash son irreversibles, son la opción recomendada para almacenar contraseñas.
- En algunos casos se combina cifrado simétrico con una función de hash.



Fuente de la imagen:
<https://criptomo.com/que-es-un-hash>

Algoritmos Hash

- Algunos de los algoritmos de hash más utilizados hoy en día son MD5, SHA1, SHA2 (formado por SHA-224, SHA-256, SHA-384, SHA-512), bcrypt, scrypt o PBKDF2. Estos tres últimos son los que se consideran los más seguros mientras que MD5 y SHA1 se consideran los más vulnerables por ataques de colisión..
- A la hora de procesar contraseñas, se busca que el algoritmo de hash sea lo más lento posible para que un atacante pueda realizar el menor número de intentos posible por unidad de tiempo.



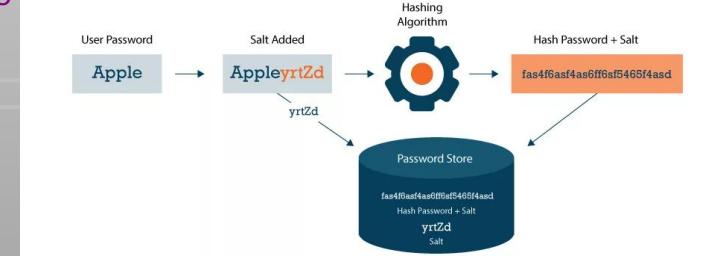
Rainbow Tables

- Teóricamente, si un ataque consigue obtener la base de datos de contraseñas y éstas están codificadas con una función hash segura, la información obtenida no se podrá utilizar para nada.
- No obstante, con la técnica de hash básica, el sistema será vulnerable a ataques basados en diccionarios de hash (Rainbow Tables).
- Estos diccionarios contienen grandes cantidades de códigos hash junto con la palabra que los ha originado, por lo tanto
- cuanto más sencilla es una contraseña, más probabilidades hay de que esté contenida en una de estas tablas.

Hash + Salt

- Para mitigar los ataques por Rainbow Tables, las técnicas de hash se combinan con fragmentos aleatorios (salt) que se concatenan a la contraseña para generar siempre un hash que no sea fácilmente predecible.
- Combinando la técnica de hash con la de salt, el atacante tendría que generar una tabla diferente para cada salt y sólo podría capturar una contraseña por cada tabla.

Hash + Salt



- Antes de almacenar la contraseña, ésta se concatena con un fragmento de datos que suele ser aleatorio.
- La cadena de datos resultante se procesa a través de la función hash.
- Y finalmente se almacenan, tanto el hash resultado como el salt.
- El proceso de verificación de una contraseña (por ejemplo, durante la autenticación) realiza las siguientes acciones:
 - En primer lugar obtiene el salt generado en el momento de guardar la contraseña.
 - A continuación, se concatena la contraseña y el salt. A la cadena resultante se le aplica la función de hash.
 - Y en último lugar compara el hash con el valor que está almacenado en la base de datos.

PBKDF2

- PBKDF2 es un algoritmo de hash muy utilizado.
- Este algoritmo permite especificar un salt y también el número de iteraciones que realizará para calcular el hash final.
- Las iteraciones se ejecutan sobre el algoritmo de hash que se usa por debajo (al menos 10000). Se suele combinar con HMAC-SHA que es una forma especial de usar SHA.



Bcrypt

- **Bcrypt** también es una de las funciones de hash más utilizadas y está diseñada para ralentizar ataques de fuerza bruta.
- En su diseño incluye un factor de trabajo que es configurable (conocido con el nombre de **rondas** o **rounds** en inglés)
- Cuanto mayor sea este valor, mayor será el tiempo que el algoritmo tarda en calcular el hash.
- En la mayoría de las implementaciones, bcrypt añade de forma automática un salt aleatorio en cada invocación

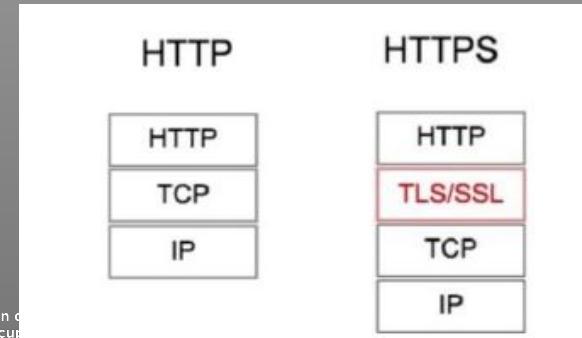
Transmisión

- La información de autenticación no debería transmitirse a través de un medio inseguro.
- Muchos protocolos de nivel de aplicación transmiten las contraseñas en texto claro, por ejemplo, **HTTP** envía los datos de los formularios sin ningún tipo de cifrado.
- En otras ocasiones son transmitidos en base64 u otros algoritmos que no presentan la seguridad suficiente.



HTTP VS HTTPS

- En muchos casos se suele utilizar una capa intermedia de cifrado.
- En el caso de HTTP, se recomienda utilizar siempre HTTPS para la transmisión de información confidencial.
- HTTPS añade una capa de cifrado por encima del protocolo TCP (TLS).
- El cifrado de datos lo veremos más adelante.





04

Vulnerabilidades de Autenticación y autorización



Broken Authentication

Broken Authentication o **Autenticación Rota** se produce cuando un sistema de autenticación es vulnerable, permitiendo a atacantes acceder a cuentas sin credenciales válidas.

Consecuencias

- Suplantación de identidad.
- Acceso no autorizado a cuentas.
- Fugas de datos confidenciales.



Fuente de la imagen:
<https://www.wallarm.com/what/broken-user-authentication>

Broken Authentication

Las vulnerabilidades pertenecientes al conjunto de Broken authentication son debidas a:

- Uso de contraseñas débiles o predecibles.
- Almacenamiento inseguro de credenciales (texto plano o cifrados débiles).
- Implementación incorrecta de autenticación multifactor (MFA).

A continuación vemos algunos ataques:



Fuente de la imagen:<https://www.authgear.com/post/broken-authentication-what-is-it-and-how-to-prevent-it>

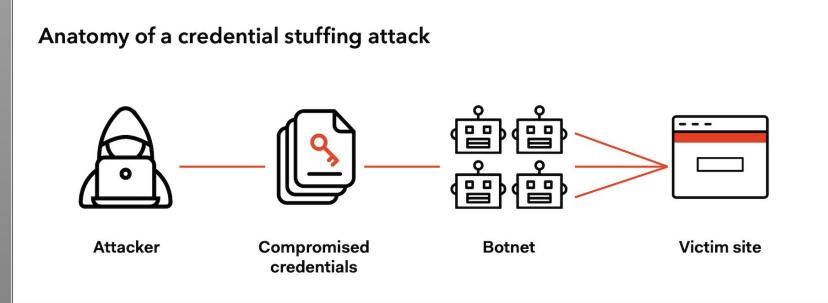
Ejemplos de Ataque – Fuerza Bruta y Credential Stuffing

Fuerza Bruta

- Un atacante prueba múltiples combinaciones de contraseñas hasta encontrar la correcta.
- Ejemplo de herramienta: **Hydra**, **Burp Suite**, **John The Ripper**.

Credential Stuffing

- Uso de credenciales filtradas de otros sitios para acceder a cuentas.



Fuente de la imagen:
<https://auth0.com/blog/what-is-credential-stuffing/>

Ejemplos de Ataque – Fuerza Bruta y Credential Stuffing

Ejemplo de ataque:

- Usuario usa la misma contraseña en varios sitios.
- Atacante obtiene credenciales filtradas y accede a otros servicios.

Mitigación:

- Imponer políticas de contraseñas seguras (longitud mínima, complejidad).
- Implementar autenticación multifactor (MFA).
- Usar CAPTCHA y bloquear intentos de inicio de sesión sospechosos.

Ejemplo de Ataque – Enumeración de Usuarios

¿Cómo funciona?

- Un atacante prueba direcciones de correo electrónico en una página de inicio de sesión y detecta qué usuarios existen según la respuesta del servidor.

Mitigación:

- Usar respuestas genéricas en el login ("Credenciales incorrectas").
- Limitar intentos fallidos de inicio de sesión.

OWASP Top 10

El boletín OWASP Top-10 agrupa las vulnerabilidades en la autenticación con las vulnerabilidades en el manejo de la sesión en una sola entrada. Ocupan la 2a posición tanto en el informe de 2013 como en el de 2017, pero el 7o en 2021.

- Explotabilidad: altamente explotables a través de ataques de fuerza bruta y/o ataques basados en diccionarios.
- Prevalencia: bastante alta debido al mal diseño e implementación de los sistemas de autenticación.
- Detectabilidad: se pueden detectar de forma manual para posteriormente explotarlos utilizando herramientas automáticas basadas en diccionarios.
- Impacto técnico: alto, sobre todo si las cuentas que se comprometen son las de administración.



CWE

Algunas entradas del CWE relacionadas con vulnerabilidades en la protección de credenciales son las siguientes:

- CWE-256: Unprotected Storage of Credentials.
- CWE-319: Cleartext Transmission of Sensitive Information.
- CWE-312: Cleartext Storage of Sensitive Information.
- CWE-257: Storing Passwords in a Recoverable Format.
- CWE-522: Insufficiently Protected Credentials.
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password.

CAPEC

Algunas entradas del CAPEC relacionadas con vulnerabilidades en la protección de credenciales son las siguientes:

- CAPEC-566: Dump Password Hashes
- CAPEC-50: Password Recovery Exploitation
- CAPEC-20: Encryption Brute Forcing
- CAPEC-49: Password Brute Forcing
- CAPEC-37: Retrieve Embedded Sensitive Data





05

Vulnerabilidad es en gestión de Sesiones



Manejo de la sesión

Las vulnerabilidades en el manejo de la sesión no afectan tanto al robo de credenciales como al robo de sesiones de usuario una vez que éste ya se ha autenticado

En este capítulo recordaremos tanto algunos de los métodos: **JWT** y **OAuth - OpenID** y también algunas de las vulnerabilidades y ataques relacionados con el manejo de la sesión.

JSON Web Token (JWT)

Veíamos que JSON Web Token (JWT) es un formato de token utilizado para autenticación basada en tokens y que tiene los siguientes componentes:

Componentes

- Header: Algoritmo de firma y tipo de token.
- Payload: Información del usuario (claims).
- Signature: Firma digital con clave secreta o clave pública/privada.

Ejemplo de JWT

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}.  
{  
  "sub": "user123",  
  "role": "admin"  
}.  
SIGNATURE
```

Riesgos de JWT y Cómo Mitigarlos

Riesgos

- Uso de algoritmos débiles (None, HS256 sin clave segura).
- JWT sin expiración permite ataques de reutilización.
- Exposición de información sensible en el payload.

Mitigación

Usar RS256 en vez de HS256 para mayor seguridad.

Incluir expiración (exp) en el token.

No almacenar información sensible en el payload.

OAUTH Y AUTENTICACIÓN FEDERADA

Veíamos que OAuth es un protocolo de autorización que permite a los usuarios acceder a recursos sin compartir contraseñas y es usado por Google, Facebook, GitHub, etc. para autenticación con terceros.

Flujos de Oauth:

- Authorization Code Flow: Más seguro, usa redirecciones y tokens temporales.
- Implicit Flow: Menos seguro, entrega tokens directamente al cliente.

Diferencias entre OAuth y OpenID Connect

OAuth 2.0

- Enfoque en autorización.
- Permite acceso a recursos sin compartir credenciales.

OpenID Connect (OIDC)

- Extiende OAuth con autenticación.
- Incluye información del usuario (id_token).

Casos de uso:

- OAuth: Permitir que una app acceda a Google Drive.
- OIDC: Permitir login en una app con la cuenta de Google.

Session Hijacking

- Por medio de un secuestro de sesión (Session Hijacking), un atacante logra obtener una cookie de sesión válida de un usuario.
- Por lo tanto, al obtener una cookie de sesión de un usuario, es posible ejecutar acciones en su nombre.



Fuente de la Imagen:
https://developer.mozilla.org/en-US/docs/Glossary/Session_Hijacking

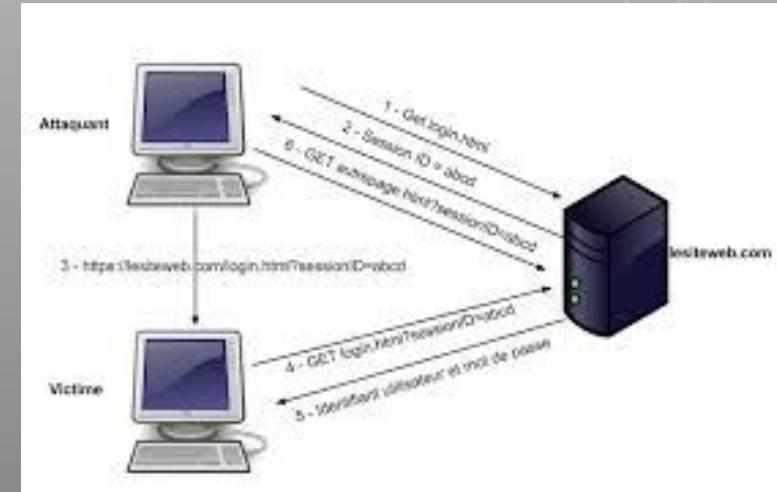
Session Hijacking

Esa cookie de sesión se puede obtener explotando alguna de las siguientes vulnerabilidades:

- Transmisión de la cookie en claro a través de medios inseguros.
- Fijación de la sesión.
- Inyección de JavaScript.

Session Fixation

La vulnerabilidad de fijación de la sesión (Session Fixation), es una variante de secuestro de sesión, en la que el atacante consigue que la víctima se autentique utilizando un identificador de sesión que el propio atacante ha generado.



Fuente de la imagen:
https://commons.wikimedia.org/wiki/File:Fixation_de_session.png

Session Fixation

En el ejemplo de la figura anterior, la secuencia de acciones que han de producirse para que el atacante consiga una cookie de sesión válida son:

1. El atacante accede a la página de autenticación.
2. El servidor crea un identificador de sesión y se lo asigna a esa sesión.
3. El atacante envía a la víctima un enlace (por ejemplo a través de un correo electrónico) que de alguna manera incluye una URL con el identificador de sesión que ha obtenido previamente.
4. La víctima utiliza el enlace para autenticarse. En el proceso de autenticación, envía el identificador de sesión que le ha enviado el atacante.
5. La víctima proporciona sus credenciales y la sesión pasa a ser una sesión autenticada .
6. El atacante conoce el identificador de sesión, que ahora pertenece a la víctima y con ese identificador, puede realizar acciones en su nombre.

Session Fixation

La forma más sencilla de ejecutar este ataque es enviando el identificador de sesión en la URL y conseguir que sea la propia víctima, la que acceda de manera voluntaria a esa dirección.

Existen también otros mecanismos más sofisticados:

- Enviar el identificador de sesión en un campo oculto de un formulario
- Enviar el identificador de sesión a través de una cookie

Para ello, se podría combinar con alguna de las técnicas que se han analizado en los capítulos anteriores, por ejemplo, inyección de JavaScript:

http://acme.com/<script>document.cookie="sessionid=863F3D316";</script>

***http://acme.com/<meta http-equiv=Set-Cookie
content="sessionid=863F3D316">***

URL Rewriting

- En ciertas circunstancias, muchos servidores envían el identificador de sesión en la URL, por ejemplo, cuando el navegador no soporta cookies:
http://www.acme.com/account.html;jsessionid=863F3D316
- Esto propicia que el identificador de sesión sea altamente vulnerable a distintos tipos de sustracción.
- La cabecera HTTP Referer contiene información sobre la página web desde la que se origina una petición, por lo tanto, si el atacante consigue redirigir el navegador de la víctima hasta un sitio web que él controla, puede capturar la URL de origen a través de esa cabecera.
- Si esa URL contiene el identificador de sesión, el atacante lo podrá utilizar para ejecutar peticiones en nombre de la víctima.

URL Rewriting

- En el siguiente ejemplo, el atacante ha conseguido insertar un enlace a su sitio web (por ejemplo, usando inyección de JavaScript)
- En el momento en que la víctima hace click en el enlace, le enviará el identificador de sesión en la cabecera Referer.



URL Rewriting

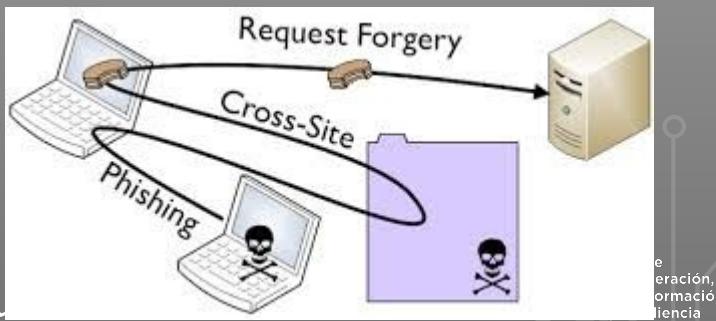
- En el ejemplo anterior, es la propia víctima la que accede, a través del enlace, al sitio web del atacante. Esta navegación también podría ejecutarse de forma automática a través de inyección de JavaScript
- En general, deberían tratarse los identificadores de sesión como si fuesen contraseñas.
- Al colocarlos en la URL, son susceptibles de otro tipo de problemas:
 - Es frecuente que los servidores web escriban en ficheros de log todas las URL, con lo que quedarán registrados también los identificadores de sesión
 - El identificador es visible en la barra del navegador. Cualquier persona próxima lo podría memorizar o sacarle una foto.

CROSS-SITE REQUEST FORGERY (CSRF)

Cross-Site Request Forgery o Falsificación de solicitudes entre sitios (CSRF) es un ataque que fuerza al usuario autenticado a ejecutar acciones no deseadas en un sitio web.

Ejemplo: Un atacante envía un enlace malicioso a un usuario autenticado que realiza una transferencia bancaria sin su consentimiento.

El hecho de utilizar el navegador web de la víctima para realizar la petición HTTP, conlleva que esta petición va a añadir de manera automática la cookie con el identificador de sesión de la víctima.



Fuente de la imagen:

<https://curiosidadesdehackers.com/cross-site-request-forgery-csrf-que-esejemplos-de-usoque-medidas-mitigadoras-podemos-usar/>



Financiado por
la Unión Europea
NextGenerationEU



MINISTERIO
DE EDUCACIÓN,
FORMACIÓN PROFESIONAL
Y DEPORTES



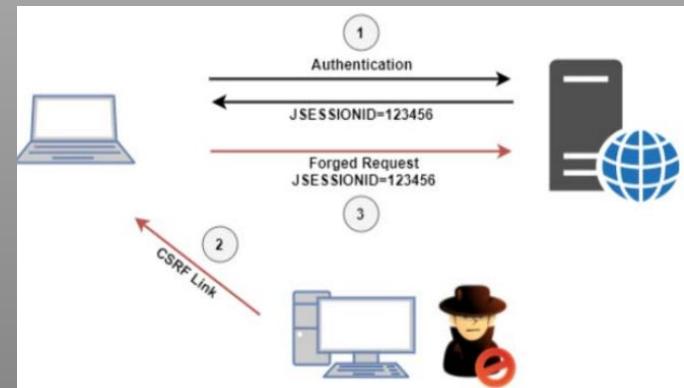
Consecuencias de CSRF

- Ejecución de acciones no autorizadas: Un usuario autenticado puede ser forzado a realizar una acción sin su consentimiento (como transferencias bancarias o cambios de contraseña).
- Cambio de configuraciones críticas: Un atacante puede modificar correos electrónicos, contraseñas o permisos en la cuenta de la víctima.
- Escalada de privilegios en sistemas mal configurados: Si un administrador es víctima de un ataque CSRF, un atacante podría otorgarse permisos elevados.
- En muchas ocasiones CSRF utiliza inyección de Java script XSS como paso previo.

Ejemplo típico de CSRF

La siguiente figura, ilustra un ejemplo típico de CSRF:

1. El usuario se autentica en el sitio web vulnerable.
2. El atacante envía un enlace al usuario (por ejemplo a través de un correo electrónico)
3. El usuario accede al enlace y a través de éste, se ejecuta una acción maliciosa sobre el sitio web vulnerable. Esta acción incluye la cookie de sesión obtenida en el primer paso.



CSRF Login

- Una variante de CSRF es CSRF Login. A través de esta vulnerabilidad, el atacante consigue abrir una sesión en una cuenta diferente a la de la víctima (por ejemplo, a través de una petición POST que simula la autenticación).
- La víctima no se da cuenta de que la cuenta en la que está operando no es la suya y realiza alguna acción que puede llegar a provocar algún daño, por ejemplo, añadir una tarjeta de crédito.
- Al igual que el ataque CSRF clásico, esta variante se basa en que el formulario de autenticación es conocido y repetible.
- Este tipo de ataques suelen transmitirse, de manera habitual, a través de enlaces dentro de correos electrónicos.

Corrección y mitigación de CSRF

- Utilizar Tokens anti-CSRF:
 - Generar un token aleatorio y único por sesión/usuario.
 - Incluirlo en formularios y peticiones sensibles (generalmente como campo oculto).
 - Validar el token en el servidor en cada petición.
- Configurar las cookies de sesión con el atributo **SameSite**.
- Verificar el origen, revisando las cabeceras **Origin** o **Referer** en peticiones sensibles y aceptando las que vienen del origen legítimo.
- Utilizar frameworks con CSRF integrada(**Laravel** o **Symfony** en PHP, **Django** o **Flask** en Python o **Spring** o **Apache Siro** en Java.

Corrección y mitigación de Gestión de Sesiones

Cookies seguras:

- Atributo **HttpOnly**: Evita acceso desde JavaScript.
- Atributo **Secure**: Solo permite envío por HTTPS.
- Atributo **SameSite**: Previene ataques CSRF.

Regeneración de ID de sesión:

- Cambiar el identificador de sesión después de autenticación para evitar Session Fixation.

Expiración y cierre de sesión:

- Establecer tiempo de expiración corto en sesiones inactivas.
- Forzar cierre de sesión en múltiples dispositivos si se detectan accesos sospechosos.

Políticas del mismo origen.

- La política de mismo origen (**Same-origin Policy**) se implementa en el navegador web y define una serie de reglas para restringir cómo un documento (perteneciente a un dominio determinado) o sus scripts, pueden interactuar con otros recursos localizados en dominios diferentes.
- Dos páginas tienen el mismo dominio si el protocolo, el nombre de dominio y el puerto son el mismo.

Políticas del mismo origen.

- Las escrituras entre dominios, por lo general, están permitidas: Enlaces, redirecciones y envío de formularios
- Las lecturas entre dominios, por lo general, no están permitidas: **XMLHttpRequest** y **Fetch API**.
- Se permite incrustar los siguientes elementos de diferentes dominios:
 - **JavaScript** con atributo **src**.
 - **Hojas de estilo** con atributo **href**.
 - **Imágenes** (PNG, JPEG, SVG, ...).
 - **Multimedia**: etiquetas video y audio.
 - **Plugins**: etiquetas **object**, **embed**, **applet**.
 - **Fuentes** a través de **@font-face** (depende del navegador).
 - **Frames e IFrames excepto** cuando está definida la cabecera **X-Frame-Options**.

Políticas del mismo origen SameSite

La **Cookie SameSite** controla cuándo se envía una cookie en peticiones cross-site (cuando el origen de la petición es distinto al dominio de la cookie). Esto mitiga ataques como CSRF.

Valores posibles:

- **Strict:** la cookie solo se envía si la petición viene del mismo sitio. Máxima seguridad, pero puede romper logins desde enlaces externos.
- **Lax:** se permite enviar la cookie en navegación top-level segura (ej.: seguir un enlace), pero no en peticiones de terceros invisibles (formularios POST, iframes). Es el valor recomendado si quieres equilibrio.
- **None:** se permite enviar la cookie en contextos de terceros, pero obliga a usar Secure (solo HTTPS).

Ejemplo: ***Set-Cookie: PHPSESSID=abc123; HttpOnly; Secure; SameSite=Lax***

CORS (Cross-Origin Resource Sharing)

Es una política de seguridad del navegador para controlar qué dominios externos pueden hacer peticiones a tu servidor y qué cabeceras/respuestas se permiten compartir.

- **Sin CORS:** el navegador bloquea solicitudes AJAX/fetch desde otro dominio hacia tu API.
- **Con CORS:** configuras qué orígenes están autorizados.

Ejemplo de configuración **CORS seguro:**

Access-Control-Allow-Origin: https://app.midominio.com

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: GET, POST

CORS (Cross-Origin Resource Sharing)

Cabeceras clave:

- **Access-Control-Allow-Origin:** indica qué dominio puede acceder (ej.: <https://app.ejemplo.com> o * si público, aunque esto último es inseguro con cookies).
- **Access-Control-Allow-Credentials:** true: permite enviar cookies o credenciales.
- **Access-Control-Allow-Methods:** qué métodos HTTP se permiten (GET, POST, PUT, DELETE).
- **Access-Control-Allow-Headers:** qué cabeceras puede enviar el cliente.

CSRF en OWASP Top 10

- **CSRF** tenía su propia entrada en el boletín OWASP Top-10 del año 2013, ocupando la 8a posición.
- En el boletín del año 2017 no aparece dentro de las 10 primeras, pero se hace referencia a ella como una vulnerabilidad que ha sido encontrada en el 5% de las aplicaciones analizadas.
- El motivo de que no se encuentre entre las 10 vulnerabilidades más importantes, se atribuye a que muchos frameworks web ya incluyen protección de serie contra CSRF.
- En el caso del resto de ataques: url rewriting, Sesión Fixation, etc. cada una pertenece a una categoría diferente:

Ataques en OWASP Top 10

- En el caso del resto de ataques: url rewriting, Sesión Fixation, etc. cada una pertenece a una categoría diferente:

Ataque	OWASP Top 10 2021 categoría	Posición / Peso
CSRF	A01: Broken Access Control	#1 (máxima)
JSON/Auth insegura	A07: Identification & Authentication Failures	#7
URL Rewriting	A07: Identification & Authentication Failures	#7
MITM	A02: Cryptographic Failures	#2 (muy alto)

Ataques de sesión CWE y CAPEC

Ataque	CWE relacionados	CAPEC relacionados
CSRF	CWE-352, CWE-613	CAPEC-62
JSON/Auth insegura	CWE-287, CWE-285, CWE-345, CWE-347	CAPEC-111, CAPEC-115, CAPEC-231, CAPEC-233
URL Rewriting	CWE-598, CWE-522, CWE-384	CAPEC-39, CAPEC-20
MITM	CWE-300, CWE-319, CWE-295, CWE-287	CAPEC-94, CAPEC-94.001, CAPEC-111

CSRF en CWE y CAPEC

Las entradas CWE relacionadas con CSRF son:

- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Los patrones de ataque relacionadas son:

- CAPEC-94: CSRF (Cross-Site Request Forgery)
- CAPEC-243: HTTP Request Smuggling
- CAPEC-591: HTTP Parameter Pollution

Nos encontramos con más de 9000 entradas en la NVD relacionadas con el CWE-352

<https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-352&resultType=records>

Ataques de sesión CWE y CAPEC

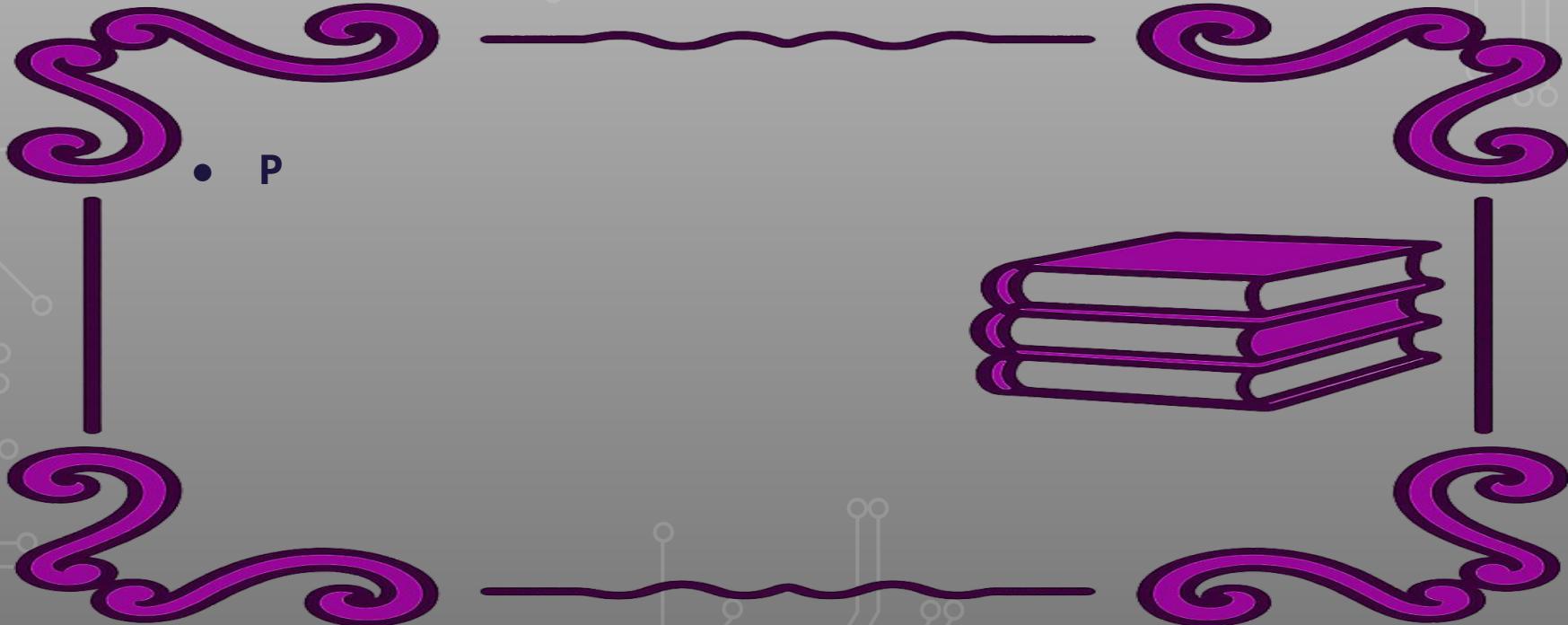
Ataque	CWE relacionados	CAPEC relacionados
CSRF	CWE-352, CWE-613	CAPEC-62
JSON/Auth insegura	CWE-287, CWE-285, CWE-345, CWE-347	CAPEC-111, CAPEC-115, CAPEC-231, CAPEC-233
URL Rewriting	CWE-598, CWE-522, CWE-384	CAPEC-39, CAPEC-20
MITM	CWE-300, CWE-319, CWE-295, CWE-287	CAPEC-94, CAPEC-94.001, CAPEC-111

Bibliografía y Webgrafía

- **Presentaciones de Puesta en Producción Segura.** *Rafael López García.*
- **Seguridad de aplicaciones.** José Losada Pérez.
- **Presentaciones de Puesta en Producción Segura.** *Rafael Fuentes Ferrer.*
- <https://owasp.org/Top10/es/>
- <https://nvd.nist.gov/vuln>



Bibliografía y Webgrafía



JUNTA DE EXTREMADURA



Plan de
Recuperación,
Transformación
y Resiliencia



Financiado por
la Unión Europea
NextGenerationEU



MINISTERIO
DE EDUCACIÓN,
FORMACIÓN PROFESIONAL
Y DEPORTES



Gracias!

¿Alguna pregunta?



informatica.iesvalledeljerteplasencia.es



coordinacion.cenfp@iesvp.es



C/ Pedro y Francisco González, s/n
10600, Plasencia (Cáceres)



927 01 77 74

