

Puesta en Producción Segura

Unidad 3.

**Protección de datos sensibles,
Control de Acceso y Errores en
la Configuración de Seguridad.**



*"En cada búsqueda apasionada
cuenta más la persecución que el
objeto perseguido"*

—El Tao del Jeet Kune Do” (1975), Bruce Lee

Objetivos

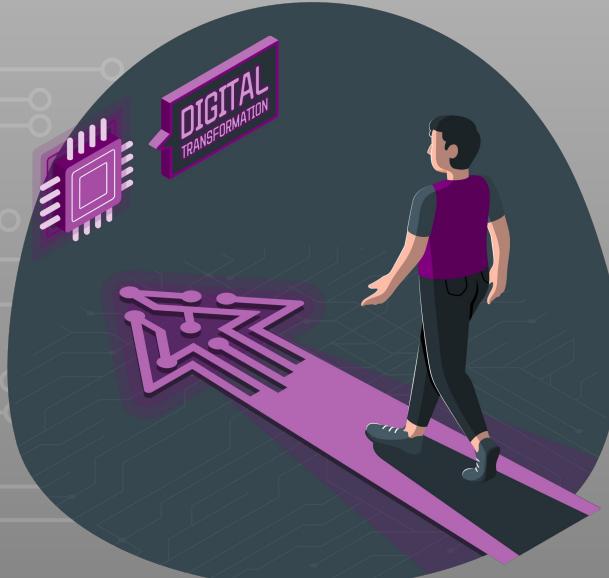
- Aprender cómo proteger datos sensibles utilizando criptografía, protocolos de seguridad y modelos de control de acceso.
- Comprender la importancia del cifrado de los datos en tránsito y conocer los protocolos a utilizar.
- Conocer las diferentes contramedidas para mitigar o corregir ataques contra el control de acceso y confidencialidad de los datos.



Contenidos

- 01** Introducción a la protección de datos sensibles.
- 02** Transmisión segura: TLS y cifrado de datos
- 03** Modelos de control de acceso

- 04** Vulnerabilidades de control de acceso
- 05** Contramedidas



01

Introducción a la protección de datos sensibles.



¿Qué son los datos sensibles?

Los datos sensibles es información confidencial que debe ser protegida contra accesos no autorizados. Por ejemplo:

- **Datos personales:** Nombre, dirección, teléfono.
- **Credenciales:** Contraseñas, tokens de sesión.
- **Datos financieros: Tarjetas de crédito, cuentas bancarias.**

La filtración de esta información suele producirse por:

- Exposición de datos sin cifrado.
- Fugas de información en tráfico de red.
- Almacenamiento inseguro en bases de datos.

Exposición de datos sensibles

- Las vulnerabilidades de **Exposición de Datos Sensibles** abarca un amplio espectro de ataques que van desde el robo de credenciales, la ejecución de ataques de intermediario ("man in the middle") y cualquier otro tipo de captura de información sin cifrar mientras ésta transita por la red.
- Algunas de las vulnerabilidades que se suelen explotar para obtener información sensible del sistema o del usuario son las siguientes:
 - Envío de información a través de comunicaciones sin cifrar: robo de cookies, credenciales, etc.
 - Uso de algoritmos de encriptación débiles que se pueden romper con ataques de fuerza bruta:
 - • Algoritmos de hash: MD5 y SHA1
 - • Algoritmos de cifrado con clave simétrica: DES
 - – Información expuesta por la propia aplicación, por ejemplo, como consecuencia de mensajes de error.

Información expuesta por la propia app

Algunos ejemplos de información sensible que exponen las propias aplicaciones son los siguientes:

- Trazas de error en Java.
- Mensajes de error de SQL.
- Mensajes de error de PHP.

Entre la información que se puede llegar a exponer:

- Información de librerías o frameworks.
- Información del sistema operativo.
- Información sobre el sistema de ficheros.
- Información sobre el código fuente de la propia aplicación.



Protocolos y sistemas que no debemos usar

Algunos de los protocolos/algoritmos inseguros que no debemos de usar son:

- Protocolos de seguridad en transmisión SSLv2, SSLv3 — obsoletos (vulnerables).
- Protocolo TLS 1.0 y TLS 1.1 — ya no se consideran seguros.
- Algoritmo de cifrado simétrico RC4 — puro roto (sesiones recuperables).
- Algoritmos de hashings MD5, SHA-1 — rotos para firmas/huellas (no usar para TLS PRFs/firmas).
- Algoritmos de cifrado DES, 3DES (DES-EDE) — clave corta / ataques por cumpleaños; 3DES está obsoleto.
- Ciphers NULL, EXPORT, IDEA, RC2(los Ciphers con métodos para codificar información para proteger su confidencialidad).
- Claves RSA key-exchange (cuando se usa sin ECDHE/DHE (métodos de intercambio de firmas)) — no ofrece PFS (Perfect Forward Secrecy es una propiedad de sistemas criptográficos que asegura que el compromiso de una clave a largo plazo no comprometa las claves de sesión de comunicaciones anteriores, generando una clave de sesión única para cada sesión de usuario).
- Cifras CBC mal implementadas en versiones antiguas (evitar usar TLS 1.0/1.1 y preferir AEAD). El CBC (Cipher-Block Chaining) es un modo de cifrado por bloques que encadena los bloques de texto plano y cifrado mediante operaciones XOR para hacerlos dependientes unos de otros.
- Claves RSA demasiado cortas (<2048 bits) o curvas débiles.



02

Transmisión segura: TLS y cifrado de datos

¿Qué es TLS?

TLS (Transport Layer Security) es un protocolo criptográfico que protege la confidencialidad, integridad y autenticidad de los datos transmitidos por la red.

Es el estándar actual que reemplazó a SSL.

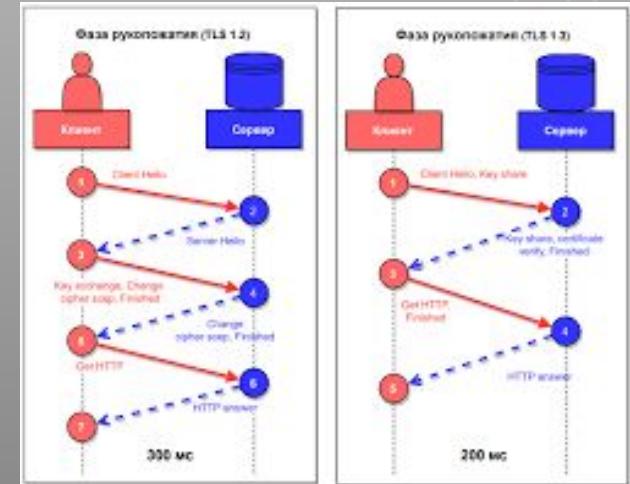
-  Cifrado → los datos viajan encriptados (no se pueden leer en claro).
-  Integridad → se detectan modificaciones o manipulaciones en tránsito.
-  Autenticación → el servidor (y opcionalmente el cliente) se identifican mediante certificados digitales.

Un ejemplo es cuando accedemos a un sitio web con <https://>, el navegador está usando TLS para establecer una conexión segura entre tu dispositivo y el servidor.

Cómo funciona TLS

Proceso de negociación (TLS Handshake)

1. Cliente y servidor intercambian versiones de TLS y algoritmos soportados.
2. Se genera una clave de sesión mediante intercambio de claves (RSA, ECDH).
3. Se establece un canal cifrado con AES-GCM o ChaCha20-Poly1305.



Fuente de la imagen:
https://commons.wikimedia.org/w/index.php?title=TLS_in_OSI.png

¿Para qué usamos los certificados?

Los **certificados digitales** se usan en TLS para **demostrar la identidad** de las partes que se comunican y establecer confianza. **Funciones principales:**

1. **Autenticación:** El certificado prueba que el servidor (o cliente, en mTLS) es quien dice ser.
2. **Confianza:** → Está firmado por una **Autoridad de Certificación (CA)** reconocida, lo que evita que un atacante suplante al servidor.
3. **Negociación segura:** Durante el *handshake TLS*, el certificado se usa para intercambiar claves de forma segura.
4. **Integridad:** Garantiza que la clave pública usada en la conexión no ha sido alterada.

Ejemplo: cuando entras a <https://banco.com>, el navegador valida el certificado del banco con una CA de confianza. Si no coincide, te muestra una advertencia.

Buenas prácticas de configuración TLS

- Deshabilitar TLS 1.0 y 1.1 (Obsoletas).
- Usar certificados confiables (Let's Encrypt, DigiCert).
- Forzar el uso de HTTPS con HTTP Strict Transport Security (HSTS).
- Usar TLS 1.2 (Estándar actual) o TLS 1.3 (Más seguro y eficiente, elimina cifrados débiles).



JUNTA DE EXTREMADURA



Introducción al cifrado y AES

¿Qué es el cifrado?

Técnica para proteger datos transformándolos en un formato ilegible sin una clave adecuada.

Tipos de cifrado

- Simétrico: Usa la misma clave para cifrar y descifrar (AES).
- Asimétrico: Usa un par de claves pública y privada (RSA, ECC).



Fuente de la imagen:
<https://commons.wikimedia.org/wiki/File:CriptografiaAsimetrica.png>

Cifrado AES

- **Advanced Encryption Standard (AES)** es el cifrado simétrico más seguro y utilizado.
- Algoritmos disponibles: AES-128, AES-192, AES-256.
- Uso en aplicaciones web, bases de datos y discos cifrados.
- Se utiliza habitualmente en los protocolos VPN.

Fuente de la imagen:

<https://www.pandasecurity.com/es/mediacente/r/cifrado-aes-guia/>

JUNTA DE EXTREMADURA



Plan de
Recuperación,
Transformación
y Resiliencia

CIFRADO AES 101



💡 Un método de cifrado que utiliza múltiples rondas de transposición, sustitución y mezcla.

💡 Hay tres tipos : AES-128, AES-192 y AES-256.

💡 Considerado el estándar de la seguridad de los datos comerciales y personales.

Funcionamiento de AES

Proceso de cifrado AES

- División del mensaje en bloques de 128 bits.
- Transformaciones matemáticas sobre los bloques usando la clave.
- Modos de operación: ECB (Electronic CodeBook), CBC (Cipher Block Chaining), GCM (Galois/Counter Mode). Ver más en <https://whitestack.com/es/blog/cifrado-aes/>

Ejemplo de cifrado en Python

```
from Crypto.Cipher import AES  
import os  
key = os.urandom(32) # Clave AES-256  
cipher = AES.new(key, AES.MODE_GCM)  
ciphertext, tag = cipher.encrypt_and_digest(b"Mensaje secreto")
```

Corrección y mitigación

- Nunca almacenar claves en código fuente (HardCode): usar almacenes keystores.
- Renovar y monitorizar certificados; usar certificados con 90 días (Let's Encrypt) o según política.
- Usar TLS (HTTPS, TLS 1.3 preferente; mínimo TLS 1.2).
- Forjar Perfect Forward Secrecy (PFS) (usar ECDHE/X25519).
- Usar cifrado autenticado (AEAD): AES-GCM, (en vez de AES-CBC), ChaCha20-Poly1305.
- Validación estricta de certificados (verificar CA, hostname, revocación/OCSP).
- HSTS en aplicaciones web, redirigir HTTP a HTTPS.
- Evitar cifrados/algoritmos inseguros o débilmente configurados.

Corrección y mitigación

- Preferir ECDSA o RSA con claves grandes ($\text{RSA} \geq 3072$ bits si se usa RSA para firmar el certificado).
- Dentro de organizaciones, implementar mTLS si necesitas autenticar clientes (mutual TLS).
- Registro y monitoreo de fallos TLS (detectar cert expirados/errores de cadena).
- Probar con herramientas de escaneo TLS (externas) tras configuración.
- Activar OCSP stapling (Técnica que permite a un servidor web entregar una respuesta OCSP (Online Certificate Status Protocol) directamente a los clientes, como los navegadores web, durante el intercambio SSL).



03

Modelos de control de acceso



Control de acceso

Una vez superada la autenticación y autorización, las **directivas de control de acceso** tienen como misión garantizar que los usuarios sólo puedan acceder a aquellas funcionalidades a las que tienen autorización.

Las **consecuencias** de este tipo de vulnerabilidades pueden ser:

- Acceso a cuentas de usuario sin producirse ninguna autenticación.
- Acceso a las funcionalidades del administrador por parte de usuarios sin permisos para ello.
- Acceso a recursos no permitidos (ficheros de disco, etc).



Principio del mínimo privilegio (POLP)

El control de acceso debe seguir el **principio de menor privilegio (Principle Of Least Privilege o POLP)**.

Este concepto consiste en limitar los permisos de acceso de un usuario a los mínimos imprescindibles para que éste pueda realizar su trabajo.



Problemas de control de acceso

Algunos tipos de problemas en el control de acceso que se estudiarán en este capítulo son los siguientes:

- Modificaciones de las URLs cuando hacen referencia a las claves primarias de los recursos.
- Manipulación de metadatos, como por ejemplo, cookies o tokens JWT
- Acceso al sistema de ficheros (atravesando directorios o permitiendo al usuario especificar el nombre del fichero).
- Redirecciones no controladas.



Modelos de Control de Acceso lógico

Para implementar el principio del mínimo privilegio dentro del control de acceso lógico, tenemos diferentes modelos de control de acceso:

- Control de acceso discrecional (**DAC**).
- Control de acceso obligatorio (**MAC**).
- Control de acceso Basado en Roles (**RBAC**).
- Basado en Atributos (**ABAC**).
- **Modelos híbridos** (combinación de varios modelos).

Más info:

<https://www.deletetechnology.com/blog/control-de-accesos-en-ciberseguridad-inform%C3%A1tica-tipos-y-ejemplos>



Control de acceso discrecional (DAC)

En el **control de acceso discrecional (DAC)**, el propietario del sistema (o archivo) define quién puede acceder y con qué permisos. Es común en sistemas operativos tradicionales, pero tiene vulnerabilidades cuando se otorgan permisos demasiado amplios.



Control de acceso obligatorio

El **control de acceso obligatorio (MAC)** impone restricciones en base a la sensibilidad del recurso y la autorización del usuario, típicamente utilizado en entornos gubernamentales o militares. Su implementación es compleja pero altamente segura.



Control de Acceso Basado en Roles (RBAC)

- En el Control de Acceso Basado en Roles (RBAC), se otorgan permisos según el rol del usuario dentro de la organización.
- Por ejemplo, un contador no puede ver registros médicos.

Ventajas e inconvenientes:

- Fácil de implementar en sistemas con roles bien definidos.
- Como norma general es flexible y escalable, ideal para empresas medianas y grandes.
- Si los permisos son muy específicos aumenta la dificultad en el escalado.

Control de Acceso Basado en Atributos (ABAC)

Utiliza múltiples atributos (rol, ubicación, hora, dispositivo) para conceder acceso. Es útil en entornos con acceso remoto o híbrido, pero requiere una gestión avanzada de políticas.

Ventajas y desventajas:

- Más flexible y dinámico que RBAC.
- Más complejo de implementar y administrar.



04

Vulnerabilidades de control de acceso



Vulnerabilidades de control de acceso

Algunos tipos de problemas en el control de acceso que se estudiarán en este capítulo son los siguientes:

- Modificaciones de las URLs cuando hacen referencia a las claves primarias de los recursos.
- Manipulación de metadatos, como por ejemplo, cookies o tokens JWT.
- Acceso al sistema de ficheros (atravesando directorios o permitiendo al usuario especificar el nombre del fichero).
- Redirecciones no controladas.



Modificaciones de URLs

- Cuando no existe control de acceso o éste es muy deficitario, conociendo la URL, el atacante puede acceder a algunas zonas de la aplicación a las que no debería.
- Un ejemplo sería utilizar el ataque de modificación de parámetros o Parameter Tampering, visto ya, para modificar la url de acceso a la administración del sitio, p.e. <http://www.acme.com/site/admin>
- Si la aplicación no hace ninguna validación adicional y el atacante conoce la URL, podrá acceder al contenido.

Modificaciones de URLs

En otro caso si la URL para acceder a la información del usuario 12345 es la siguiente (ese número representa la clave primaria en la tabla de usuarios): <http://www.acme.com/users/12345/profile>, y la aplicación no valida que un usuario sólo pueda acceder a la información de su propia cuenta, es posible acceder a la cuenta de otro usuario, simplemente cambiando el identificador numérico en la URL



Modificaciones de cookies y tokens JWT

- En algunos casos, la aplicación almacena los roles de usuario en una cookie o en un token JWT.

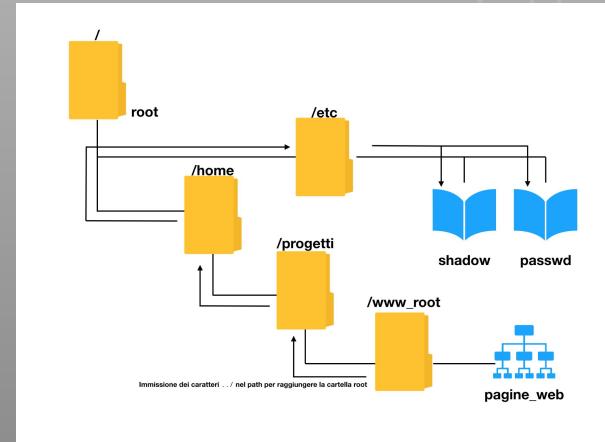
```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: JSESSIONID=12345
Set-Cookie: user_roles=user,admin
```

- Si no se realiza ningún tipo de validación sobre esa información (por ejemplo, a través de una firma digital) es posible que el atacante consiga realizar modificaciones y cambiar los roles del usuario.



Path traversal

- Ya vimos en el capítulo de inyección el ataque que Atravesar directorios (más conocido por su nombre en inglés **Directory Traversal o Path Traversal**), es una variación de **Parameter tampering** que permite a un atacante acceder a ficheros y directorios a los que no debería.
- Como vimos, se pueden atravesar directorios modificando URLs, parámetros de formularios, cookies, etc.



Fuente de la imagen:
https://commons.wikimedia.org/w/index.php?title=File:Directory_traversal.png

Path traversal

- El siguiente ejemplo, permite acceder al fichero de contraseñas de Unix atravesando directorios a través del valor de una cookie.
- En el fragmento de código PHP, se concatena el valor de la cookie para obtener la ruta a una plantilla que está almacenada en el sistema de ficheros.

```
<?php
$template = 'user.php';
if ( isset( $_COOKIE['TEMPLATE'] ) )
    $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/php/templates/" . $template );
?>
```



Path traversal

- El atacante modifica la cookie para que apunte a una localización fuera del directorio de la aplicación.
- Y consigue atravesar directorios y obtener el
- contenido del fichero de contraseñas de Unix.

```
GET /home.php HTTP/1.0  
Cookie: TEMPLATE=../../../../etc/passwd
```

```
HTTP/1.0 200 OK  
Content-Type: text/html  
Server: Apache Tomcat  
  
root:aSCVhb01:0:1:root:/bin/bash  
...
```

Redirecciones

Se produce una **redirección en una petición HTTP** cuando el servidor envía en la respuesta:

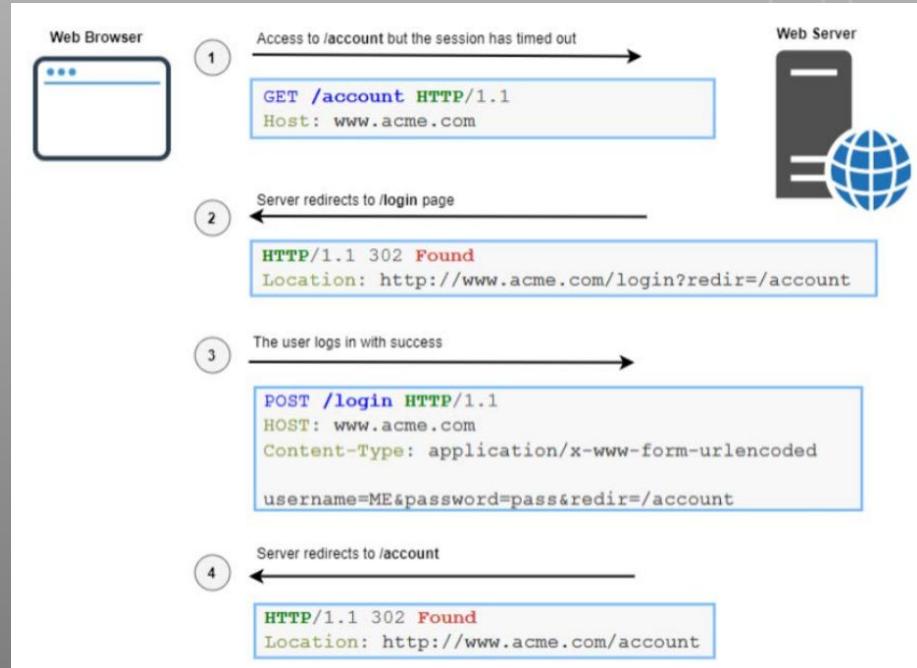
- Un código de respuesta 3xx.
- Una cabecera **Location** con la URL a la que el cliente web realizará una navegación de forma automática.

Se produce una **redirección incontrolada** cuando:

- La URL de la redirección es un parámetro de entrada de la app.
- El servidor web realiza la redirección a la URL sin validar previamente que esa URL es legítima.

Redirecciones

- El siguiente ejemplo, muestra una secuencia de peticiones muy habitual a la hora de realizar el proceso de autenticación.
- Esta secuencia incluye varias redirecciones



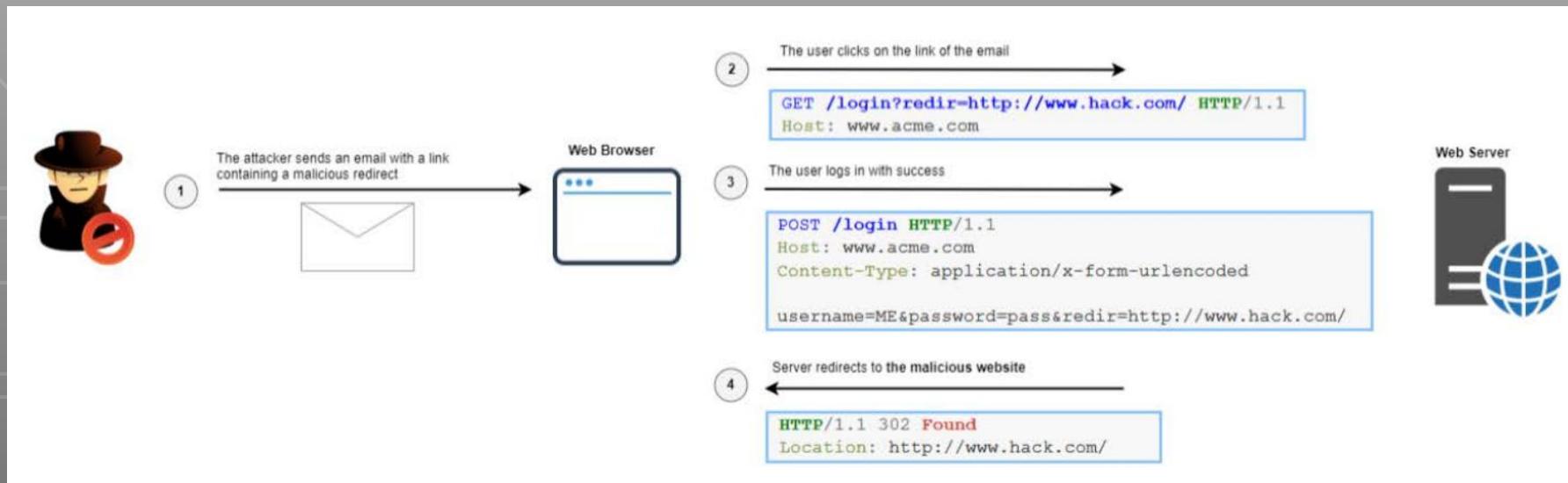
Redirecciones

1. El usuario intenta conectarse a su cuenta pero la sesión ha caducado y el servidor no le permite acceder.
2. En la respuesta, el servidor envía una redirección a la página de autenticación, con un parámetro de autenticación "redir" que apunta a la URL a la que el usuario intentaba acceder.
3. El usuario se autentica y el parámetro "redir" se mantiene en el POST que se genera al enviar el formulario.
4. Tras la autenticación, el servidor envía otra redirección a la URL en la que el usuario puede consultar su cuenta.



Redirecciones

Si la aplicación no valida el parámetro “redir”, podría darse el siguiente escenario:



Redirecciones

1. El atacante envía un email a la víctima con un enlace que contiene una redirección maliciosa.
2. La víctima hace click en el enlace del email.
3. Enlace lleva al usuario a la página de autenticación.
4. Una vez que se ha autenticado, es el propio servidor web el que redirige al usuario al sitio malicioso.
5. Ese sitio web malicioso puede ser una “copia” del original que intenta capturar información del usuario, por ejemplo, podría volver a pedirle las credenciales
6. O puede intentar explotar alguna otra vulnerabilidad, como CSRF para realizar una petición maliciosa en la cuenta del usuario o Reescritura de URL y capturar la sesión del usuario a través de la cabecera *Referer*.

Redirecciones

- Con los forwards sucede lo mismo que con las redirecciones.
- En el caso del forwarding de peticiones, se le envían los parámetros de la petición a la URL de destino.
- Las principales diferencias entre redirect y forward son:
 - El forward se resuelve internamente en el servidor web, es transparente al usuario. La redirección produce una nueva petición a través del código 3xx y la cabecera *Location*.
 - El *forward* envía los parámetros de la petición a la nueva URL

Corrección y mitigación de problemas de control de acceso

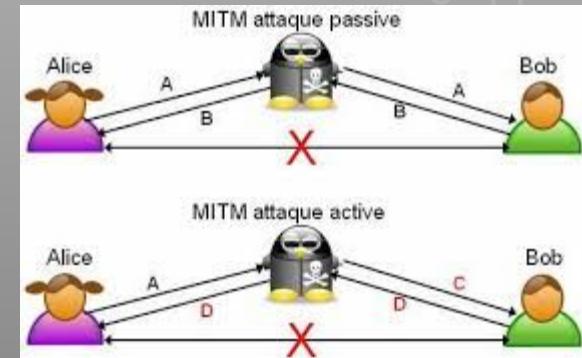
- Implementar métodos de control de acceso.
- Por defecto utilizar política de mínimo privilegio.
- La política por defecto debe ser la denegación de acceso.
- Los controles de acceso deben implementarse en el servidor
(Importante:los controles de acceso y validaciones en la vista NO son una medida de prevención).
- Se debe imponer la propiedad de "dueño" para que un usuario sólo pueda acceder a "sus" registros.
- Se deberían incluir las pruebas de control de acceso, como un elemento más del flujo del control de calidad (QA).

Corrección y mitigación de problemas de control de acceso

- Siempre que sea posible, **evitar redirecciones**.
- Si no es posible evitarlas, intentar que éstas **no utilicen entradas de usuario**.
- Para validar las URL, siempre que sea posible, se debe utilizar una **lista blanca de valores válidos**.
- Una buena práctica consiste en no utilizar un parámetro con la dirección real y en su lugar, **emplear una clave almacenada internamente en un mapa**.
- Utilizar una **página intermedia que requiera la confirmación** del usuario antes de realizar la redirección.

Man-in-the-middle

En el ataque de intermediario (man-in-the-middle), el atacante es capaz de interferir en las comunicaciones entre un cliente y un servidor, capturando los mensajes entre ambos interlocutores e incluso inyectando mensajes modificados en el canal de comunicación.



Fuente de la imagen:
[https://commons.wikimedia.org
/wiki/File:Attaque_Man_In_The_Middle.jpg](https://commons.wikimedia.org/wiki/File:Attaque_Man_In_The_Middle.jpg)

Man-in-the-middle

- Con los ataques de intermediario, es posible ejecutar secuestros de sesión (**Session Hijacking**).
- Otros métodos similares de secuestrar la sesión con técnicas de intermediario:
 - **Sniffing**: captura de tráfico de red
 - **Sidejacking**: también consiste en capturar paquetes de datos, con el objetivo de obtener cookies de sesión. Se suele utilizar en redes WI-FI abiertas.
 - **Evil Twin**: consiste en crear una red WI-FI con el mismo nombre que la red legítima y conseguir que el usuario se conecte. El atacante consigue actuar de intermediario.

Man in the middle en OWASP Top 10

Esta vulnerabilidad ocupa la 3a posición en el boletín OWASP Top-10 del año 2017. En 2021 cambió a Cryptographic Failures y está de 2a,

- Explotabilidad: ataques basados en diccionarios, captura de información en claro transmitida por un medio no seguro, uso de algoritmos débiles de cifrado o hash, etc.
- Prevalencia: ataque de alto impacto en los últimos años.
- Detectabilidad: variable en función de la debilidad, para transmisiones por medios no seguros se puede hacer de forma automática.
- Impacto técnico: puede llegar a ser muy elevado.



Man in the middle en CWE

Algunas entradas del CWE relacionadas con esta vulnerabilidad son las siguientes:

- CWE-200: Information Exposure
- CWE-201: Information Exposure Through Sent Data
- CWE-205: Information Exposure Through Behavioral Discrepancy
- CWE-532: Information Exposure Through Log Files
- CWE-203: Information Exposure Through Discrepancy
- CWE-209: Information Exposure Through an Error Message

- CWE-538: File and Directory Information Exposure
- - CWE-219: Sensitive Data Under Web Root
- CWE-311: Missing Encryption of Sensitive Data
- CWE-214: Information Exposure Through Process Environment
- CWE-540: Information Exposure Through Source Code.
- CWE-598: Information Exposure Through Query Strings in GET Request

Man in the middle en CAPEC

Algunas entradas del CAPEC relacionadas con esta vulnerabilidad son las siguientes:

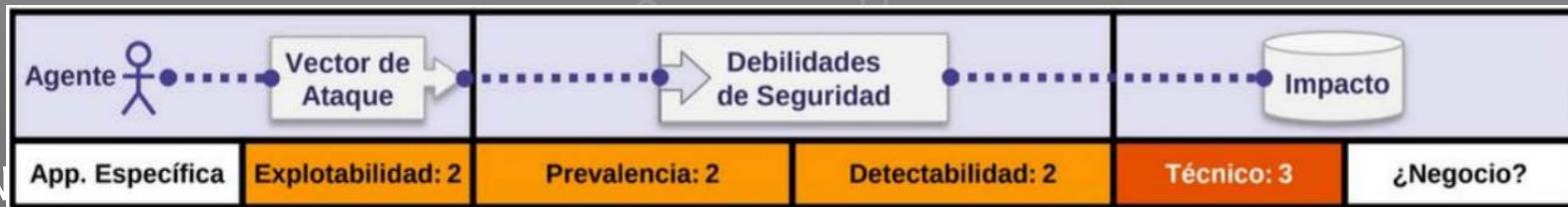
- CAPEC-116: Excavation
- CAPEC-117: Interception
- CAPEC-169: Footprinting
- CAPEC-22: Exploiting Trust in Client
- CAPEC-472: Browser Fingerprinting
- CAPEC-216: Communication Channel Manipulation
- CWE-212: Improper Cross-boundary Removal of Sensitive Data



Control de acceso en OWASP Top 10

Las vulnerabilidades en el control de acceso ocupan la 1a posición en el boletín OWASP Top-10 del año 2021

- **Explotabilidad:** las herramientas automáticas podrían detectar si no hay control de acceso, pero no pueden determinar si las reglas son correctas.
- **Prevalencia y detectabilidad:** como no es sencillo de detectar de forma automática y es frecuente que las pruebas no cubran este tipo de comprobaciones, la prevalencia es alta.
- **Impacto técnico:** puede llegar a ser alto, por ejemplo, si las funcionalidades del administrador no están correctamente protegidas.



Control de acceso en CWE

Algunas entradas del CWE relacionadas con el control de acceso son las siguientes:

- CWE-284: Improper Access Control
- CWE-283: Unverified Ownership
- CWE-282: Improper Ownership Management
- CWE-601: URL Redirection to Untrusted Site ('Open Redirect')
- CWE-23: Relative Path Traversal
- CWE-35: Path Traversal: '.../...//'
- CWE-36: Absolute Path Traversal.
- CWE-37: Path Traversal: '/absolute pathname/here'
- CWE-39: Path Traversal: 'C:dirname'

Control de acceso en CAPEC

Algunas entradas del CAPEC relacionadas con el control de acceso son las siguientes:

- CAPEC-225: Subvert Access Control.
- CAPEC-180: Exploiting Incorrectly Configured Access Control Security Levels
- CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs
- CAPEC-233: Privilege Escalation
- CAPEC-597: Absolute Path Traversal
- CAPEC-139: Relative Path Traversal
- CAPEC-76: Manipulating Web Input to File System Calls



05

Contramedidas: Captcha, HTST, CSP



Contramedidas

Tenemos un conjunto de herramientas, métodos que nos van a ayudar evitar los ataques de control de acceso, confidencialidad de datos, etc. Entre ellos:

- Captcha, recaptcha.
- Cabeceras http de seguridad.



CAPTCHA

Los **CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart)** son otro mecanismo que se usa mucho en ciberseguridad, pero a diferencia de las directivas HTTP (que refuerzan la seguridad del navegador/servidor), los CAPTCHAs se centran en diferenciar usuarios humanos de bots automatizados.

En resumen: **mitigan abusos de automatización** que un simple control de acceso no detiene.

Ventajas: frenan bots, protegen recursos, mitigan abuso.

Inconvenientes: afectan a la usabilidad, pueden ser molestos, y los más simples hoy en día ya no son efectivos frente a IA.

CAPTCHA

Evitan que bots automáticos realicen acciones masivas o maliciosas, como:

- Creación de cuentas falsas.
- Ataques de **fuerza bruta** contra formularios de login.
- Envío masivo de spam en formularios de contacto o comentarios.
- Acaparar entradas/tickets (scalping).
- Raspado de datos (web scraping) en exceso.

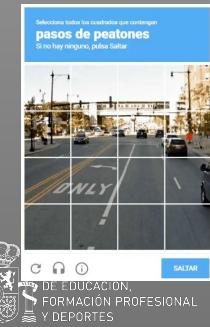
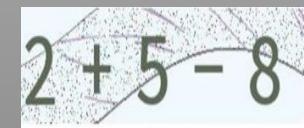
¿Qué ataques ayudan a prevenir?

- Automatización maliciosa (bots de spam, registros masivos).
- Credential stuffing (uso de listas de credenciales filtradas).
- Brute force (intentos repetitivos de login).
- Denegación de servicio a nivel aplicación (formularios explotados por bots).



Tipos de CAPTCHA

1. CAPTCHA de texto / imágenes distorsionadas.
 - Muestran letras o números con distorsión.
 - El usuario debe transcribirlos.
 - Simples, pero hoy en día son fáciles de resolver con IA/OCR.
2. CAPTCHA de preguntas lógicas / aritméticas
 - Ej: "¿Cuánto es $3 + 5$?".
 - Fácil para humanos, barato de implementar.
 - Fácil de romper con bots programados.
3. CAPTCHA de imágenes / selección de objetos
 - Ej: "Selecciona todas las imágenes con semáforos".
 - Muy usado en Google reCAPTCHA v2.
 - Más difícil de automatizar.
 - Puede ser molesto para el usuario.



CAPTCHA VS RECAPTCHA

La principal diferencia es que **CAPTCHA** es el **término general para las pruebas que diferencian humanos de bots**, mientras que **reCAPTCHA** es un sistema específico desarrollado por Google que representa una **evolución avanzada** de los CAPTCHAs. reCAPTCHA utiliza **algoritmos más sofisticados y análisis de comportamiento** para verificar al usuario, llegando incluso a ofrecer una experiencia "invisible" donde el desafío no se presenta si el usuario se comporta de forma normal.

Tipos de CAPTCHA

4. reCAPTCHA v2 (Google)

- Opción "No soy un robot" (checkbox).
- Analiza el comportamiento (movimiento del ratón, clics, cookies).
- Más usable.
- Si sospecha, te manda al reto de imágenes.

5. Invisible reCAPTCHA / reCAPTCHA v3

- El usuario no ve nada, se analiza su comportamiento en la navegación (tiempo de permanencia, interacciones, reputación IP, etc.).
- Muy cómodo, sin fricción.
- Dependencia de Google y posible invasión de privacidad.

6. hCAPTCHA (alternativa a reCAPTCHA)

- Similar al de Google, pero más centrado en privacidad.
- Algunas webs lo prefieren porque no depende de Google.



Cabeceras de seguridad http

Tenemos un conjunto de cabeceras HTTP de seguridad que se **configuran en el servidor** (Apache, Nginx, etc.) y ayudan a mitigar ataques relacionados con control de acceso, integridad, confidencialidad y exposición de datos:

HSTS: protege la confidencialidad y fuerza HTTPS.

CSP: protege contra XSS y control de recursos.

X-Frame-Options / frame-ancestors: protege contra clickjacking.

X-Content-Type-Options: protege contra MIME sniffing.

Referrer-Policy: protege contra fuga de información.

Permissions-Policy: controla qué APIs puede usar el navegador.

HSTS (HTTP Strict Transport Security)

- Obliga al navegador a comunicarse solo por HTTPS y nunca volver a intentar HTTP.
- Un atacante no podrá forzar al usuario a cargar la web en HTTP inseguro.

*Strict-Transport-Security: max-age=31536000;
includeSubDomains; preload*



X-Frame-Options / frame-ancestors (CSP)

- Evita que tu web se incruste en un <iframe>.

X-Frame-Options: DENY

- También se puede implementar en CSP (lo veremos más adelante) con:

Content-Security-Policy: frame-ancestors 'none'



X-Content-Type-Options

- Evita que el navegador intente "adivinar" el tipo de archivo.

X-Content-Type-Options: nosniff



Referrer-Policy

- Controla qué información de la URL previa se envía al navegar a otra página.
Referrer-Policy: no-referrer
- Otras menos estrictas:
no-referrer-when-downgrade, origin,
origin-when-cross-origin, same-origin, strict-origin,
strict-origin-when-cross-origin, unsafe-url
- Evita fuga de datos sensibles en parámetros de URL.



Permissions-Policy

- Limita el acceso a APIs del navegador (cámara, geolocalización, etc.).
Permissions-Policy: geolocation=(), camera=(), microphone=()
- Mitigación: Reduce superficie de ataque y abuso de permisos.



Content Security Policy (CSP)

- **Content Security Policy (CSP)** es un mecanismo que permite restringir los contenidos que el navegador puede cargar en un sitio web.
- Su principal funcionalidad es la de detener ataques de inyección de código y en especial XSS.
- CSP se suele especificar en una **cabecera** dentro de la **respuesta HTTP** que el servidor envía al navegador.
- Aunque también se puede definir en un **tag <meta>** dentro del documento HTML.

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://; child-src 'none';">
```

Content Security Policy (CSP)

El valor de CSP está formado por una serie de directivas con la siguiente sintaxis:

Content-Security-Policy: <policy-directive>; <policy-directive>;

- Existen directivas para especificar restricciones en:
 - Restricciones en el origen de los datos.
 - Restricciones en el documento.
 - Restricciones en la navegación.
 - Directivas de notificación de contenido que no cumple con la política definida.



Ejemplos CSP

- La siguiente directiva restringe el origen de los datos (scripts, hojas de estilo, imágenes, iframe, etc.) al propio sitio web. Por lo tanto, no es posible cargar ningún elemento de un sitio web externo.

Content-Security-Policy: default-src 'self'



Ejemplos CSP

La siguiente directiva, por defecto, también restringe el origen de los datos al propio sitio web, pero a continuación se especifica que:

- Es posible cargar imágenes de cualquier origen (**img-src**).
- Otro contenido multimedia será accesible desde media1.com y media2.com (**media-src**).
- El origen de los scripts será userscripts.example.com (**script-src**).

*Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src userscripts.example.com*

Directiva CSP script-src

La directiva **script-src** de CSP es muy importante para controlar de manera exhaustiva el origen de todos los scripts:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src>

Su sintaxis es la siguiente:

*Content-Security-Policy: script-src <source>;
script-src <source>;*

Directiva CSP frame-ancestors

- La directiva frame-ancestors permite definir cuándo es posible incluir un sitio web dentro de un iframe:
Content-Security-Policy: frame-ancestors 'none';
Content-Security-Policy: frame-ancestors 'self https://www.example.org;'
Similar a la cabecera X-FRAME-OPTIONS.

Directiva CSP report-uri y report-to

- Con las directivas report-uri y report-to, es posible especificar una URL a la que se enviarán notificaciones cuando el navegador detecte contenido que no cumple con la política definida:
Content-Security-Policy: default-src https://; report-uri /csp-violation-report-endpoint/
- Y a esta URL se enviará un mensaje JSON con la siguiente información:

```
1  {
2    "csp-report": {
3      "document-uri": "http://example.com/signup.html",
4      "referrer": "",
5      "blocked-uri": "http://example.com/css/style.css",
6      "violated-directive": "style-src cdn.example.com",
7      "original-policy": "default-src 'none'; style-src cdn.example.com; report-uri /_csp-reports",
8      "disposition": "report"
9    }
10 }
```



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

- <https://www.deletetechnology.com/blog/control-de-accesos-en-seguridad-inform%C3%A1tica-tipos-y-ejemplos>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>



Gracias!

¿Alguna pregunta?



informatica.iesvalledeljerteplasencia.es



coordinacion.cenfp@iesvp.es



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



927 01 77 74

