Práctica 1: Servidor Flask con cookies/sesiones, endpoint seguro vs. inseguro, despliegue con Gunicorn y NGINX (HTTP/80 & HTTPS/443

Trabajo en parejas

Curso 2025–2026

Resumen

En esta práctica (en parejas) se implementará un servicio web con **gestión de cookies/-sesiones**, un **endpoint inseguro** y otro **seguro**, **desplegado con Gunicorn** y **expuesto detrás de NGINX** por `HTTP/80` e `HTTPS/443`. Se deberá **demostrar con evidencias** el comportamiento inseguro/seguro tanto en acceso directo como a través de NGINX, y **analizar si el endpoint inseguro sigue siéndolo por el puerto 80 y si por el 443 pasa a ser seguro**. Si no es así, se deberá **proponer y justificar una solución**.

Flexibilidad tecnológica Se puede sustituir **Flask** por otra tecnología (p.ej., *FastAPI, Django, Express.js, Spring Boot, ASP.NET*) y/o **Gunicorn** por otra (p.ej., *Uvicorn, Hypercorn, Daphne*) siempre que se **argumente** la decisión en la memoria (ventajas/inconvenientes, impacto en seguridad, rendimiento y despliegue).

1. Objetivos de aprendizaje

- Comprender cookies, atributos (`Secure`, `HttpOnly`, `SameSite`) y sesiones en servidor.
- Diferenciar endpoints «inseguros» y «seguros» y justificar por qué.
- Desplegar con un servidor WSGI/ASGI (*Gunicorn* u otro) detrás de *NGINX* como *reverse proxy*.
- Configurar `HTTP (80)` y `HTTPS (443)`, TLS y directivas de seguridad (p.ej., `HSTS`).
- Diseñar y ejecutar una **batería de pruebas** reproducible (`curl`, navegador, capturas).
- Elaborar una **memoria técnica** y una **demostración en vídeo** sintética.

2. Requisitos funcionales

✓ **RF1. Servidor web** con **gestión de cookies/sesiones** (login simple de demostración).

Tener dos ficheros de configuración, incompatibles entre ellos, solo se tiene un bloque server (el 443 o el 80). En el segundo fichero de comunicación, el 443, se redirige el 80 al 443.

/inseguro --> Falta evidenciar que transmite datos en claro (HTTP) para la mei

/seguro --> CSRF no implementa

Puedes mejorar forzando redirección automática a HTTPS para evitar error 400.

RF2. **Endpoint inseguro** (p.ej., expone información sin autenticación ni TLS; o acepta/expone cookie sin `Secure`; o vulnerable a *session fixation* si justificáis) y **endpoint seguro** (p.ej., requiere sesión válida y se sirve sólo por HTTPS, `Secure`+ `HttpOnly`+ `SameSite`, CSRF si aplica).

Capturas log / curl!

RF3. **Demostraciones/evidencias** claras de por qué uno es inseguro y el otro es seguro (peticiones `curl`, capturas del navegador, *DevTools* mostrando cookies y atributos).

RF4. **Despliegue con Gunicorn** (u otra alternativa justificada) y **NGINX** como *reverse proxy*.

RF5. Exposición por `HTTP/80` y `HTTPS/443`. Analizar:

/ seguro --> accesible por HTTP → actualmente da error 400, sería mejor redirigir a HTTPS. ⚠

- ¿El endpoint inseguro sigue siéndolo por el 80 detrás de NGINX?
- ¿El endpoint «se vuelve» seguro por el 443? Si no, **proponer solución** (redirecciones, `HSTS`, `Secure` en cookies, `proxy_set_header` correcto, *scheme aware* en app, políticas NGINX, etc.).

3. Arquitectura sugerida (ejemplo con Flask) - INCOMPLETA

Aplicación mínima con endpoint inseguro/seguro

Listing 1: app.py (ejemplo con Flask)

```python
from flask import Flask, request, session, jsonify, redirect, url_for,
    make_response
from datetime import timedelta

app = Flask(__name__)
app.secret_key = "CHANGE_ME_IN_PRODUCTION"
app.permanent_session_lifetime = timedelta(minutes=20)

def cookie_flags(resp):
    # Forzar flags en cookies de sesión si el servidor está detrás de
       HTTPS
    # En desarrollo, podéis parametrizar por entorno o encabezado X-
       Forwarded-Proto.
    session_cookie = app.session_cookie_name
    if session_cookie in resp.headers.getlist('Set-Cookie'):
        # En Flask, usa SESSION_COOKIE_* en config para hacerlo global
        pass
    return resp

@app.route("/login", methods=["POST"])
def login():
    user = request.form.get("user")
    if not user:
        return jsonify({"error": "user required"}), 400
    session.permanent = True
    session["user"] = user
    resp = make_response(jsonify({"ok": True, "user": user}))
```

```
    # Alternativa: configurar en app.config: SESSION_COOKIE_SECURE=True
       , etc.
    return cookie_flags(resp)

@app.route("/logout", methods=["POST"])
def logout():
    session.clear()
    return jsonify({"ok": True})

@app.route("/inseguro")
def inseguro():
    # Información expuesta sin autenticación ni TLS garantizado
    return jsonify({
        "debug": True,
        "message": "Este endpoint no requiere sesión y puede ir por
           HTTP."
    })

@app.route("/seguro")
def seguro():
    # Requiere sesión y pretende servirse solo por HTTPS
    if "user" not in session:
        return jsonify({"error": "unauthorized"}), 401
    # (Opcional) Comprobar esquema si el proxy pasa X-Forwarded-Proto
    scheme = request.headers.get("X-Forwarded-Proto", request.scheme)
    if scheme != "https":
        return jsonify({"error": "use https"}), 400
    return jsonify({"secret": "contenido solo para usuarios
       autenticados por HTTPS"})
```

Gunicorn (ejecución local)

Listing 2: Arranque con Gunicorn

```
# Instala dependencias
pip install flask gunicorn

# Ejecuta (bind al loopback si va detrás de NGINX)
gunicorn -w 4 -b 127.0.0.1:8000 app:app
```

NGINX: HTTP (80) e HTTPS (443)

Listing 3: /etc/nginx/sites-available/practica

```
# Zona de rate limiting (opcional)
limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

server {
    listen 80;
    server_name ejemplo.local;
```

```
    # Opción A (para experimentar): permitir /inseguro por HTTP y
    # forzar redirección a HTTPS solo en /seguro
    location /seguro {
        return 301 https://$host$request_uri;
    }

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        limit_req zone=api burst=20 nodelay;
        proxy_pass http://127.0.0.1:8000;
    }
}

server {
    listen 443 ssl http2;
    server_name ejemplo.local;

    ssl_certificate     /etc/ssl/certs/ejemplo.crt;
    ssl_certificate_key /etc/ssl/private/ejemplo.key;
    include /etc/nginx/snippets/ssl-params.conf;

    # HSTS para endurecer (comentad durante pruebas si os bloquea HTTP)
    add_header Strict-Transport-Security "max-age=31536000;
       includeSubDomains" always;

    # Endurecimiento adicional
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options DENY;
    add_header Referrer-Policy no-referrer;

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

Discusión clave. A continuación se presentan diferentes hilos de discusión que se deben argumentar dentro de la memoria con evidencias.

- Si se mantiene `/inseguro` accesible por `HTTP/80`, seguirá siendo **inseguro**, incluso detrás de NGINX: se transmite en claro.
- `/seguro` con redirección a `HTTPS/443` y cookies con `Secure` se considera **seguro**, *siempre que*:

1. La app **valide** que el `scheme` es `https` (vía `X-Forwarded-Proto`) o se confía en el *reverse proxy*.
2. Las cookies de sesión se marquen `Secure`, `HttpOnly` y `SameSite=Lax/Strict`.
3. Se active `HSTS` para evitar *downgrade* a HTTP (tras primera visita HTTPS).

4. Pruebas y evidencias requeridas

Se deben incluir capturas y/o comandos reproducibles:

Comprobaciones con `curl`

```
# 1) Endpoint inseguro por HTTP (debe funcionar, mostrando por qué es
   inseguro)
curl -i http://ejemplo.local/inseguro

# 2) Endpoint seguro por HTTP (debe redirigir a HTTPS o rechazar)
curl -i http://ejemplo.local/seguro

# 3) Login por HTTPS y almacenamiento de cookie
curl -i -c cookies.txt -X POST -d "user=alice" https://ejemplo.local/
   login

# 4) Acceso a /seguro por HTTPS con cookie (debe permitir)
curl -i -b cookies.txt https://ejemplo.local/seguro

# 5) Acceso a /seguro por HTTP con cookie (debe fallar o redirigir a
   HTTPS)
curl -i -b cookies.txt http://ejemplo.local/seguro

# 6) Ver atributos de la cookie en navegador (captura DevTools >
   Application > Cookies)
```

Cualquier dns.local, se encuentra dentro del ordenador

Evidencias mínimas

- Cabeceras y respuestas de `curl` (`-i`) que prueban redirecciones, errores, o datos en claro.
- Capturas de **DevTools** mostrando cookies con `Secure`/`HttpOnly`/`SameSite`.
- Fragmentos de logs de NGINX y/o Gunicorn (fecha/hora, método, ruta, código).
- Diagrama simple de arquitectura (app → Gunicorn → NGINX → Cliente).

5. Seguridad en puertos

Si al probar el endpoint /inseguro en el puerto 80 no se comporta realmente como inseguro, o si al probar el endpoint /seguro en el puerto 443 no se comporta realmente como seguro, entonces se debe proponer y justificar una solución.

Ejemplos de diferentes propuestas:

- Redirigir todo HTTP a HTTPS salvo una ruta de laboratorio (`/inseguro`) para demostrar el riesgo.
- **Bloquear** `/seguro` en HTTP (`deny all`) o devolver `403` en HTTP.
- Forzar `Secure` en cookies, `HSTS`, y validación explícita del esquema en la app.
- Autenticación en `/seguro` (sesión/token) y **no** aceptar credenciales por HTTP.

La propuesta se tiene que implementar y argumentar.

6. Extensiones opcionales para «subir el nivel»

- **CSRF** en rutas POST y prueba de su efectividad.
- Almacenamiento de sesión en **Redis** (servidor de sesiones compartidas).
- **Docker Compose** con servicios (app, proxy, redis) y `Makefile` de tareas.
- **Certbot/Let's Encrypt** o CA local para TLS en entorno de práctica.
- Políticas de seguridad: `CSP`, `X-Content-Type-Options`, `Referrer-Policy`.
- **Rate limiting** en NGINX y **WAF** básico (p.ej., bloqueo de métodos).
- **Pruebas automatizadas** (pytest) y **CI** (lint + tests).
- **Observabilidad**: logs estructurados y métricas (*Prometheus*/`/metrics`).
- **Ataques controlados**: *session fixation*/*cookie theft* demostrados en entorno seguro.
- Sustituir **Flask/Gunicorn** por tecnología alternativa **argumentada** (p.ej., FastAPI+Uvicorn).

7. Rúbrica de evaluación (100 puntos)

Criterio	**Puntos**
Arquitectura funcional: app + Gunicorn/alternativa + NGINX funcionando (80/443)	20
Gestión de cookies/sesiones (`Secure`, `HttpOnly`, `SameSite`) correctamente aplicadas y explicadas	15
Endpoint **inseguro** demostrado y razonado (riesgo real)	10
Endpoint **seguro** demostrado y razonado (incluye validación de esquema / TLS)	15
Análisis HTTP/80 vs HTTPS/443 a través de NGINX + propuesta de solución si procede	15
Pruebas y evidencias (curl, capturas, logs) claras y reproducibles	10
Calidad de la memoria (claridad, justificación técnica, referencias)	10

Vídeo 30s presentando la app funcionando (conciso y nítido)	5
Total	**100**

8. Entregables

- **Memoria** (este documento completado con vuestro caso: decisiones, configuración final, pruebas, evidencias, conclusiones).
- **Código**: copia del **repositorio Git** con `app`, `requirements`, `gunicorn` (o alternativa), `nginx.conf`, `Dockerfile`/`compose` si procede, `scripts`.
- **Vídeo (30s)** presentando la aplicación en funcionamiento (navegación mínima + terminal con `curl` + una prueba de cookie segura).

9. Sugerencia de estructura de la memoria

1. **Introducción y objetivos.**
2. **Decisiones tecnológicas** (y alternativas descartadas).
3. **Arquitectura** (diagrama y flujo de peticiones).
4. **Implementación** (gestión de sesiones/cookies, endpoints, configuración WSGI/ASGI, NGINX).
5. **Seguridad** (amenazas, mitigaciones: TLS, HSTS, flags de cookie, CSRF, límites).
6. **Pruebas y evidencias** (comandos, capturas, resultados esperados/obtenidos).
7. **Análisis 80 vs 443 por NGINX** y **propuesta de mejora**.
8. **Conclusiones y trabajo futuro**.

10. Notas y buenas prácticas

- No subáis claves reales al repo. Usad variables de entorno.
- Documentad cómo reproducir (`README.md`, `Makefile`).
- Si cambiáis Flask/Gunicorn, **justificad** el cambio y adaptad ejemplos.
- Podéis usar certificados auto-firmados en laboratorio, explicando implicaciones.

11. Anexos

Ejemplo de `ssl-params.conf` endurecido (laboratorio)

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_session_timeout 10m;
ssl_session_cache shared:SSL:10m;
```

Systemd (opcional) para Gunicorn

Listing 4: /etc/systemd/system/gunicorn.service

```
[Unit]
Description=Gunicorn for practica1
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/srv/practica1
ExecStart=/usr/bin/gunicorn -w 4 -b 127.0.0.1:8000 app:app
Restart=always

[Install]
WantedBy=multi-user.target
```