Sesión 1: Seguridad en aplicaciones web

Duración: 45 minutos

🎯 Objetivos de aprendizaje

Al finalizar esta sesión, el participante será capaz de:

- Identificar las principales vulnerabilidades en aplicaciones web Python
- Implementar validación de entrada segura
- Configurar manejo seguro de sesiones y autenticación
- Aplicar principios de seguridad en el desarrollo web
- Reconocer y prevenir ataques comunes (SQL injection, XSS, CSRF)

📋 Contenido

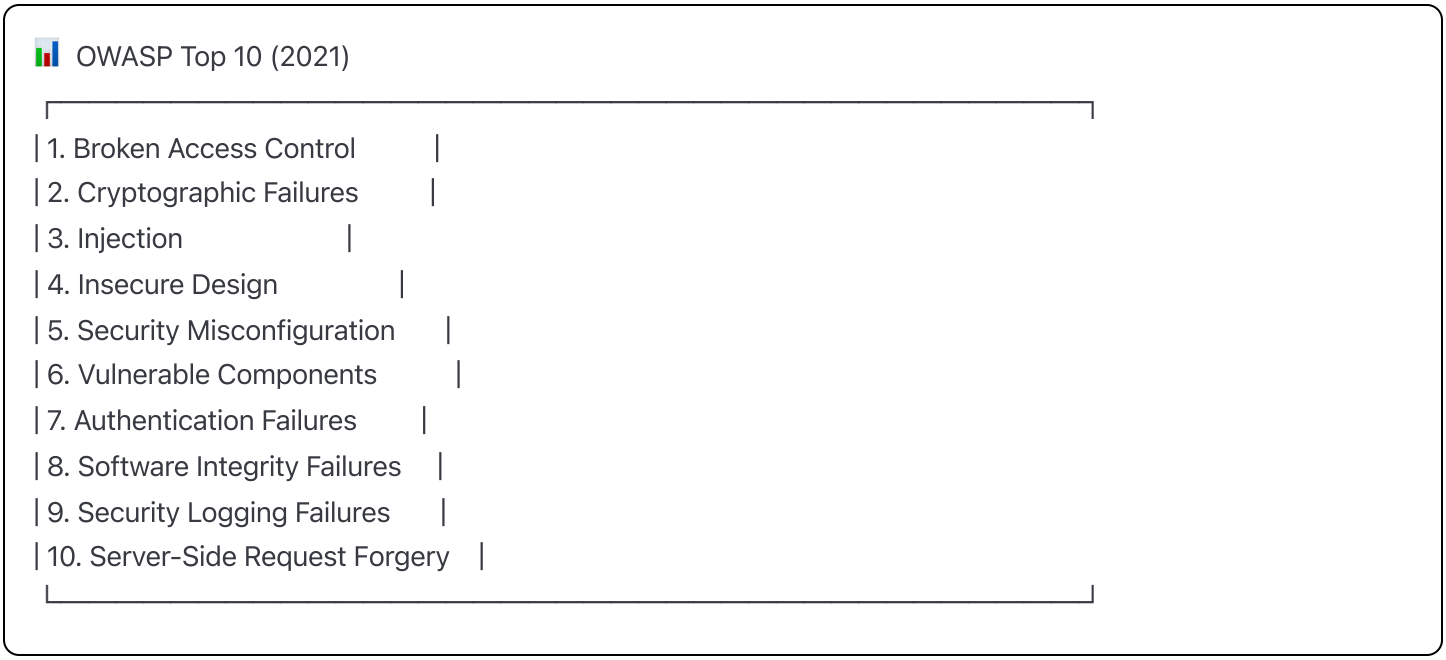
1. Introducción a la Seguridad Web (5 min)

¿Por qué es crítica la seguridad?

- Las aplicaciones web son el punto de entrada más común para ataques
- Python es ampliamente usado en desarrollo web (Flask, Django, FastAPI)
- Un error puede comprometer datos sensibles de miles de usuarios

Analogía: Una aplicación web es como una casa. Las vulnerabilidades son puertas y ventanas mal cerradas que permiten el acceso no autorizado.

2. OWASP Top 10 - Los Riesgos Más Críticos (10 min)



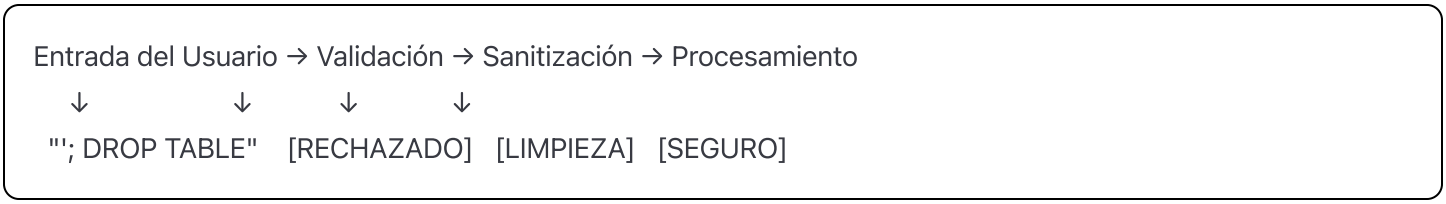
Enfoque en Python:

- **Injection:** Especialmente SQL injection y command injection
- **Broken Authentication:** Manejo inseguro de sesiones y passwords
- **XSS:** Cross-Site Scripting en templates

3. Validación de Entrada - Primera Línea de Defensa (10 min)

Principio fundamental: "Nunca confíes en datos del usuario"

Estrategias de validación:

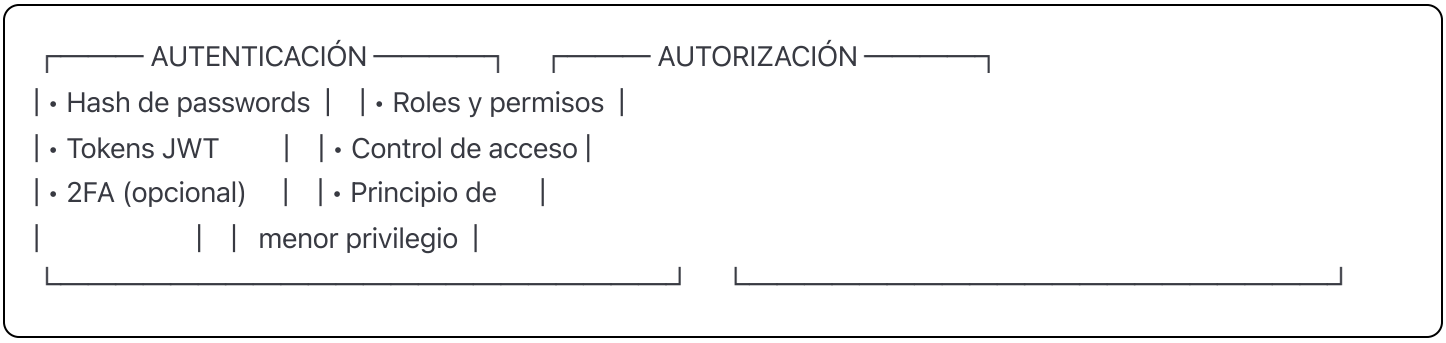


Técnicas en Python:

- Uso de librerías como `marshmallow` o `pydantic`
- Validación de tipos de datos
- Escape de caracteres especiales
- Límites de longitud y formato

4. Autenticación y Gestión de Sesiones (10 min)

Componentes de un sistema seguro:



Buenas prácticas:

- Usar `werkzeug.security` para hash de passwords
- Implementar expiración de sesiones
- Regenerar session IDs después del login
- Usar HTTPS siempre

5. Prevención de Ataques Comunes (8 min)

SQL Injection:

```python
# ❌ VULNERABLE
query = f"SELECT * FROM users WHERE id = {user_id}"

# ✅ SEGURO
query = "SELECT * FROM users WHERE id = ?"
cursor.execute(query, (user_id,))
```

XSS Prevention:

```python
# ❌ VULNERABLE
return f"<h1>Hola {username}</h1>"

# ✅ SEGURO (con escape)
return f"<h1>Hola {html.escape(username)}</h1>"
```

CSRF Protection:

- Usar tokens CSRF en formularios
- Validar referer headers

- Implementar SameSite cookies

⚠️ Advertencias Críticas

🚨 NUNCA hagas esto:

- Almacenar passwords en texto plano
- Concatenar directamente input del usuario en queries SQL
- Ejecutar comandos del sistema sin validación
- Exponer información sensible en logs o errores

🛡️ SIEMPRE implementa:

- Validación en el servidor (no solo cliente)
- Logging de eventos de seguridad
- Límites de rate limiting
- Headers de seguridad HTTP

🔧 Herramientas Esenciales

- **Bandit:** Análisis estático de seguridad para Python
- **Safety:** Verificación de dependencias vulnerables
- **OWASP ZAP:** Pruebas de penetración automatizadas
- **pip-audit:** Auditoría de paquetes Python

📝 Resumen Final

1. **La seguridad es responsabilidad de todos** los desarrolladores
2. **Validar toda entrada** del usuario sin excepción
3. **Usar librerías probadas** para criptografía y autenticación
4. **Implementar defensa en profundidad** (múltiples capas)
5. **Mantener dependencias actualizadas** y sin vulnerabilidades
6. **Documentar y registrar** eventos de seguridad

🏃 Actividad Práctica Sugerida

Ejercicio: Analiza una aplicación Flask simple e identifica 5 vulnerabilidades de seguridad. Luego implementa las correcciones necesarias usando las técnicas vistas en clase.

Tiempo estimado: 15-20 minutos adicionales (opcional para casa)