Módulo 5: Seguridad de bases de datos

Lección 1: Amenazas y Riesgos en Bases de Datos

Objetivos de la Lección

Al concluir la sesión, el estudiante será capaz de:

1. **Identificar** ataques de inyección SQL y otros vectores que provocan fugas de datos.
2. **Analizar** cómo la exposición de información sensible (PII, credenciales, datos de pago) se convierte en un riesgo legal y operativo.
3. **Evaluar** la gravedad de un incidente sobre la base de la confidencialidad, integridad y disponibilidad de los datos y proponer controles adecuados.

Introducción a la Lección

Las bases de datos son el corazón de la mayoría de las aplicaciones modernas; su compromiso se traduce en pérdidas económicas, sanciones regulatorias y daño reputacional. En esta lección examinaremos las principales amenazas—desde la inyección SQL hasta la filtración de PII—y aprenderemos a mapear cada vulnerabilidad a sus riesgos y contramedidas.

Desarrollo del Tema

Inyección de código (SQL, NoSQL, ORM Bypass, GraphQL)

Definición:
La **inyección de código** ocurre cuando un atacante logra insertar o manipular

instrucciones dentro de una consulta a la base de datos. Esto sucede cuando la aplicación **no valida ni sanitiza adecuadamente la entrada del usuario**, permitiendo que se ejecute código malicioso.

- Inyección SQL:
  - **Ejemplo de Inyección SQL:**

    ```
    # Ejemplo inseguro en Python
    usuario = input("Nombre de usuario: ")
    consulta = f"SELECT * FROM usuarios WHERE nombre = '{usuario}'"
    ```

  - **Si el atacante escribe:**

    ```
    ' OR '1'='1
    ```

  - **La consulta resultante sería:**

    ```
    SELECT * FROM usuarios WHERE nombre = '' OR '1'='1'
    ```

    (esto devolvería todas las filas con información de la base de datos exponiendo así datos confidenciales)

- NoSQL: Ocurre cuando se manipulan consultas JSON u objetos sin sanitizar.
  - **Ejemplo inseguro en MongoDB (Node.js):**
    - db.users.find({ username: req.body.user });
  - **Un atacante podría enviar:**
    - { "$ne": null }
  - **La consulta se transforma en:**
    - db.users.find({ username: { "$ne": null } }); Retorna todos los usuarios, exponiendo información
- ORM Bypass: Los ORM (como SQLAlchemy, Django ORM o Hibernate) facilitan consultas seguras, **pero no son inmunes** si se usan mal.
  - **Ejemplo inseguro en SQLAlchemy:**
    - session.execute(f"SELECT * FROM usuarios WHERE id = {user_id}")
  - **Si `user_id` proviene del usuario sin validar:**

- `user_id = "1; DROP TABLE usuarios;"` (Podría eliminar la tabla.)

- GraphQL:
  - En GraphQL, la manipulación ocurre si el usuario puede alterar la estructura de las consultas.
  - Ejemplo:

```
query {
  users {
    id
    name
    password
  }
}
```

  - Si no se controla, el atacante podría acceder a campos sensibles como contraseñas o tokens.

Vectores adicionales

Tipo de motor	Payload típico	Ejemplo de daño
NoSQL (MongoDB)	`{"$gt": ""}` en parámetro JSON	El atacante inserta un operador lógico de MongoDB ($gt = “greater than”), lo que hace que la consulta devuelva **todos los documentos** sin aplicar filtros → **robo de información**.
GraphQL	`query { __schema { types { name } } }`	Este payload pide al servidor que **revele el esquema interno completo** de la API (tipos,

		campos, relaciones). Esa información se usa para planear ataques posteriores.
ORM bypass	`User.where("1=1")`	El atacante manipula el constructor del ORM para que siempre devuelva **todas las filas**, o incluso inyectar un **bloque SQL crudo** (como DROP TABLE usuarios;).

Técnicas de detección

- **SQLMap (comando)** con modo *tamper* para descubrir vectores ocultos.
    - **SQLMap -** es una herramienta automatizada para detectar y explotar inyecciones SQL.
    - **Modo Tamper -** los *tamper scripts* son pequeños módulos que transforman/obfuscan la carga (payload) que SQLMap envía al servidor.
- **Análisis de logs:** picos súbitos de `UNION, OR 1=1` o `$ne`
    - El **análisis de logs** consiste en revisar los registros del servidor (web, aplicación o base de datos) para **detectar patrones anómalos** en las solicitudes que llegan.
    - **Ejemplo de posible anomalía**
        1. Picos súbitos de `UNION, OR 1=1 o $ne` en los parámetros; esto significa que al revisar estos registros (logs), se detectan aumentos repentinos (picos) de ciertos patrones sospechosos que suelen indicar intentos de inyección.
            - En un periodo corto, el log muestra **muchas solicitudes** con esos patrones (por ejemplo, 30 intentos con UNION SELECT en un minuto).
- **Database Activity Monitoring (DAM)**: inspecciona tráfico SQL en red.

- **Database Activity Monitoring (DAM)** es una **técnica o herramienta de seguridad** que **inspecciona, registra y analiza en tiempo real el tráfico SQL** (consultas, comandos y respuestas) que circula hacia y desde las bases de datos.
- DAM **observa las consultas SQL que viajan por la red** (por ejemplo, de un servidor web al servidor de base de datos).
- Esto ayuda a detectar patrones sospechosos de:
    1. `UNION SELECT`
    2. `OR 1=1`
    3. Accesos a tablas sensibles (usuarios, credit_card, etc.)
    4. Transferencias de grandes volúmenes de datos.

Controles reforzados

1. **Parámetros tipados**: es una variable que se pasa a la consulta SQL de forma **segura y controlada por el motor de base de datos**, no como texto concatenado.
    a. **Ejemplo de implementación**: `cursor.execute("SELECT * FROM usuarios WHERE id = %(id)s", {"id": user_id})`
        1. **Que significa esto?**
            - El valor user_id se envía aparte del texto SQL.
            - El motor lo trata como dato, no como código.
            - Evita que un atacante inserte comandos SQL maliciosos.

2. **Cortafuegos SQL (WAF interno)**: también llamado **Web Application Firewall (WAF)** o **Database Firewall**, es una capa de protección que **filtra y analiza las consultas SQL** antes de que lleguen a la base de datos.
    a. **Reglas de configuración**:
        1. **OWASP CRS (Core Rule Set):** es un conjunto de reglas de seguridad predefinidas por OWASP que detectan patrones maliciosos (por ejemplo: `UNION SELECT`, `OR 1=1`, `DROP TABLE`).
        2. **Listas blancas de consultas:** se definen las **consultas SQL legítimas** que una aplicación puede ejecutar. Si llega una consulta diferente, el firewall la bloquea.
3. **Política de cuentas DB**: Es una práctica de **mínimos privilegios (Principle of Least Privilege)** aplicada a las cuentas que acceden a la base de datos.
    a. **Por ejemplo:**
        1. La aplicación (por ejemplo, un sitio web) solo debería leer datos (SELECT).
        2. No necesita permisos para borrar (DROP) o modificar estructuras (ALTER).
        3. Así, si la app es comprometida, el atacante no podrá destruir o cambiar la base de datos.
4. **Escaneo de infraestructura como código** (IaC): **Infraestructura como Código (IaC)** significa definir servidores, redes y configuraciones mediante **archivos automatizados** (como Terraform o Ansible).

Caso real — *British Airways 2018*
Inyección JS (Java Script) → skimmer robó datos de pago de 380 000 clientes. El root cause fue un API con consulta SQL formada por string-concat. Demanda colectiva + multa GDPR £20 M.

- ¿Qué es un **Skimmer**?
    - Un **skimmer** es un script malicioso (generalmente en **JavaScript**) diseñado para robar información **confidencial** que el usuario introduce

en una página web, como datos de tarjetas de crédito, contraseñas o direcciones.

- Este tipo de ataque se conoce como "**Web Skimming**" o "**Magecart attack**".

Información privada, protegida o sensitiva

Definición conceptual

Datos cuyo uso está regulado por leyes (HIPAA para salud, FERPA para expedientes académicos, GLBA para datos financieros). Incluye además secretos comerciales, resultados de investigación y propiedad intelectual.

Clasificación y etiquetado

- Es el proceso mediante el cual una organización **identifica, clasifica y etiqueta la información** según su **nivel de sensibilidad o confidencialidad**.
- **Esquema típico de clasificación: Public, Internal, Confidential, Restricted** (esquema típico).
- **Herramientas de descubrimiento**

Herramienta	**Función Principal**	**Ejemplo de patrón detectado**
MS Purview	Descubre y etiqueta datos sensibles en M365, Azure, SharePoint, etc.	Números de tarjeta, pasaporte, SSN.
Varonis	Analiza permisos, accesos y contenido sensible en archivos y repositorios.	Archivos con información personal o financiera.
AWS Macie	Escanea datos almacenados en S3	IBAN, correos electrónicos,

	(Amazon Cloud) y detecta información personal.	identificadores médicos (ICD-10).

Los patrones como **SSN** (Social Security Number), **IBAN** (número bancario internacional) o **ICD-10** (códigos médicos) se reconocen mediante expresiones regulares (**regex (Expresion Regular –** secuencia de caracteres que definene un patrón de búsqueda dentro de un texto**)**) y modelos preentrenados.

Controles avanzados

- **Column-Level Encryption** (CLE) o **FLE** (MongoDB Field-Level Encryption): cifrar solo atributos como `ssn` (Social Security Number).
- **Dynamic Data Masking** (DDM): **Dynamic Data Masking (DDM)** o **enmascaramiento dinámico de datos** es una técnica que **oculta parcialmente la información sensible** cuando es consultada por **usuarios con bajos privilegios**, sin modificar los datos reales almacenados en la base de datos.
    - **Ejemplo:** Frente a usuarios de bajo privilegio (`999-##-####`).
- **Redacción de consultas:** técnica donde el sistema o motor de base de datos modifica automáticamente las consultas SQL originales antes de ejecutarlas, agregando condiciones o filtros de seguridad.
    - **Ejemplo:** comando “tenant_id = :user_tenant” (comando en el cual permite que un usuario pueda ver su información especifica, no se muestra información de otros usuario (protegiendo la confidencialidad de dichos datos)).

Información identificable o deducible (PII)

PII directa vs. indirecta

- **Directa:** nombre, SSN, e-mail institucional.

- **Indirecta (quasi-identifiers):** fecha de nacimiento + código postal + sexo, puede permitir *re-identificación*.

Modelos o metodologías de protección y evaluación de privacidad

- **K-anonymity**: Es un **modelo matemático de anonimización de datos**. Busca garantizar que **cada persona en un conjunto de datos no se pueda distinguir de al menos otras *x* personas** con características similares.
- **L-diversity / T-closeness**: Asegura que, dentro de cada grupo de personas con los mismos quasi-identificadores, **haya al menos "L" valores diferentes para el atributo sensible** (como diagnóstico o salario).
- **DPIA (Data Protection Impact Assessment)**: Una **evaluación formal de impacto en la protección de datos personales**, exigida por el **Reglamento General de Protección de Datos (GDPR)** en su **artículo 35**.

Controles

- **Pseudonimización**: remplazos reversibles con tabla de correspondencia cifrada.
- **Differential Privacy**: Es una técnica matemática y estadística que permite analizar o compartir información de un conjunto de datos sin revelar información privada de ninguna persona individual. Agrega ruido estadístico para revelar información pero no información privada.
- **Tokenización**: se emite token único por usuario sin exponer identidad real.

Ejemplo técnico de controles

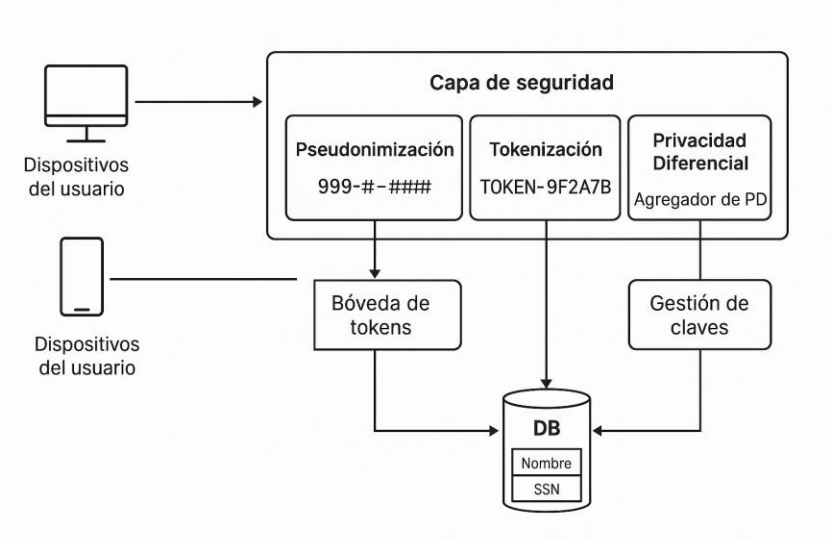

Imagen1: Ejemplo de uso de los controles para mantener la seguridad de usuario a nivel de identificación en bases de datos.

Información de pago y contraseñas

Estándares y definiciones

- **PAN** (Primary Account Number) = 16 dígitos de tarjeta de credito.
- **CVV/CVC** no debe almacenarse (*PCI DSS 3.2.1* sección 3.2). Entiendase que **PCI DSS** exige cifrado fuerte, segmentación de red y logs inmutables para datos de tarjetas.
- **Strong hash** para contraseñas: `argon2id, time_cost=3, memory=64 MiB (Mebibytes), parallelism=2`. Las contraseñas se almacenan con hash resistente.
    - **argon2id:** variante híbrida (resistente a ataques de diccionario y de canales laterales).
    - **time_cost = 3:** número de iteraciones (más alto = más lento).
    - **memory = 64 MiB:** memoria que debe usar el atacante por intento (clave para frenar GPUs).
    - **parallelism = 2:** hilos de ejecución (ajústarlo a cores disponibles).

Errores recurrentes

- Hashing con SHA-1 + sal global → vulnerable a GPU cracking.
- Almacenar claves en variables de entorno dentro del contenedor y subir la imagen a Docker Hub público.
- Reutilizar clave de cifrado para varias tablas (viola “key-separation”).

Buenas prácticas ampliadas

1. **HSM/KMS** con *envelope encryption*

a. HSM (Hardware Security Module): es un dispositivo físico seguro que almacena y protege llaves criptográficas (no pueden salir del hardware).

b. KMS (Key Management Service): es el servicio equivalente en la nube (por ejemplo, AWS KMS, Azure Key Vault, Google Cloud KMS).

2. **Tokenización PCI**:
   a. En lugar de guardar el número real, se genera un **token (alias aleatorio)**
   b. El sistema almacena **solo el token**, y **solo el servidor seguro de tokenización** puede recuperar el número real (proceso de *de-tokenización*).
   c. **Ejemplo**

```
PAN real: 4111 2222 3333 4444
Token: TKN-9835-221A-4444
```

Imagen 2. Ejemplo de Tokenicacion con el numero de una tarjeta de crédito, PAN (Primary Account Number)

3. **Rotación automática** de contraseñas: Puede ser usado en cuentas de servicio vía *AWS Secrets Manager rotation lambda*.
   a. Una cuenta de servicio es una credencial usada por aplicaciones o procesos automáticos (no por humanos).
   b. Rotación de contraseñas = cambiar contraseñas periódicamente (por ejemplo, cada 30 días).
   c. AWS Secrets Manager es un servicio de Amazon que guarda contraseñas, tokens o llaves API cifradas y permite rotarlas automáticamente.
   d. Lambda = una función sin servidor que se ejecuta automáticamente para generar y actualizar nuevas contraseñas.

4. HSTS + TLS 1.3 para transporte; forzar MFA para credenciales sensibles.

a. **HSTS (HTTP Strict Transport Security):** Una política que obliga al navegador a usar siempre HTTPS, evitando que un usuario se conecte por HTTP inseguro.

b. **TLS 1.3:** La versión más reciente del protocolo de cifrado para comunicaciones seguras en red. Mejora rendimiento y elimina algoritmos débiles.

Escenario real

- Filtración de contraseñas hashed con MD5 → rainbow tables → violación masiva de cuentas.

Relación con Otros Conceptos

- La inyección SQL conecta con el módulo de **vulnerabilidades de software**; los daños se cuantifican mediante el **análisis de riesgos** estudiado en clases anteriores.
- La protección de PII se alinea con los **principios de privacidad** y las **políticas y confianza** abordadas en el módulo G.

Resumen de la Lección

Revisamos cómo la inyección de código y la exposición de datos sensibles comprometen la seguridad de bases de datos. Aprendimos a clasificar la información (privada, PII, pagos) y a enlazar cada riesgo con controles técnicos (cifrado, hashing robusto), administrativos (políticas de retención) y normativos (PCI DSS, GDPR). Una estrategia integral combina defensa en profundidad, monitoreo continuo y cumplimiento regulatorio.

Actividad de la Lección — “Pentest y Mitigación”

1. Se entrega un contenedor Docker con una API Flask y BD SQLite vulnerable a SQLi y que almacena contraseñas en MD5.
2. En grupos de dos:
    - **Explota** la inyección para extraer la tabla `users`.
    - **Identifica** qué columnas contienen PII directa o indirecta.
    - **Aplica** un *hot-fix*: parametriza consultas y migra los hashes a Argon2id.
    - **Demuestra** en un informe (máx. 2 pág.) la diferencia entre los riesgos antes y después del parche.
3. Subir PR en GitHub; CI incorporado corre `sqlmap --batch y python -m pip-audit`.

Referencias Adicionales

- Pfleeger, C. P., Pfleeger, S. L., & Coles-Kemp, L. (2023). *Security in Computing* (6.ª ed.). Addison-Wesley.
- OWASP. *Top 10 Web 2025* y *Cheat Sheet: SQL Injection Prevention* (2025).
- NIST SP 800-122 (2020). *Guide to Protecting the Confidentiality of Personally Identifiable Information*.
- PCI SSC. *PCI DSS v4.0* (2022).
- EU GDPR (2018) — Artículos 4, 32 y 33 sobre PII y notificación de brechas.