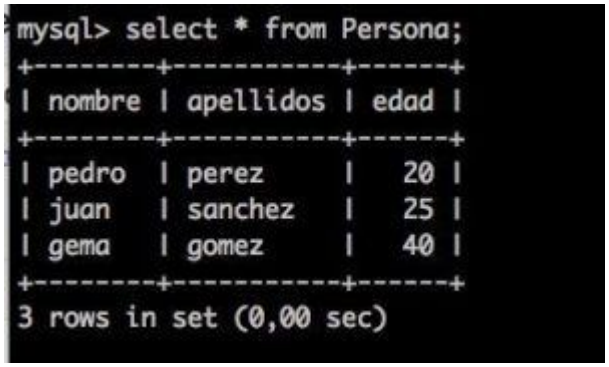
JPA SQL Injection es un problema del que muchos desarrolladores se olvidan ,ya que consideran que JPA esta totalmente protegido contra inyeccion de SQL. ¿Es esto cierto o podemos vernos afectados?. La realidad es que no estamos totalmente protegidos . Depende mucho de como se haya construido el código . Vamos a ver un ejemplo sencillo que nos ayude a entenderlo. Supongamos que tenemos una base de datos con los siguientes datos.



Vamos a construirnos un ejemplo con JPA que nos permita seleccionar datos de la tabla. El primer paso es definir las dependencias de Maven que vamos a utilizar.

```
<dependencies>

                <dependency>
<groupId>org.hibernate.javax.persistence</groupId>
                        <artifactId>hibernate-jpa-2.0-api</artifactId>
                        <version>1.0.1.Final</version>
                </dependency>

        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>5.2.10.Final</version>
```

```
        </dependency>
        <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.42</version>
</dependency>
        </dependencies><dependencies>

                <dependency>
<groupId>org.hibernate.javax.persistence</groupId>
                        <artifactId>hibernate-jpa-2.0-api</artifactId>
                        <version>1.0.1.Final</version>
                </dependency>

        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>5.2.10.Final</version>
        </dependency>
        <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.42</version>
</dependency>
        </dependencies>
```

El siguiente paso es definir la clase de Entidad que vamos a utilizar:

```
package com.arquitecturajava;
```

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {
	@Id
	private String nombre;
	private String apellidos;
	private int edad;
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Persona(String nombre, String apellidos, int edad) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
```

```
                this.edad = edad;
        }
        public Persona() {
                super();
        }
}
```

Por último nos queda ver como queda el programa principal que va a realizar una consulta:

```
package com.arquitecturajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class Principal {

        public static void main(String[] args) {

                seleccionarPersona("pedro", "perez");

        }

        private static void seleccionarPersona(String nombre, String
apellidos) {
                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
```

```
		EntityManager em = emf.createEntityManager();
		try {
			String sql = "select p from Persona p ";
			if (nombre != null || apellidos != null) {
				sql += " where ";
			}
			if (nombre != null) {

				sql +=  " nombre='" + nombre + "'";
			}
			if (apellidos != null) {

				if(nombre!=null) {
					sql += " and  apellidos='" +
apellidos + "'";
				}else{
					sql += "  apellidos='" +
apellidos + "'";
				}
			}

			TypedQuery<Persona> consulta =
em.createQuery(sql, Persona.class);

			List<Persona> lista =
consulta.getResultList();
			lista.forEach((p) -> {

				System.out.println(p.getNombre());

			});
```

```
                } catch (Exception e) {

                        e.printStackTrace();
                } finally {
                        em.close();

                }
        }
}
```

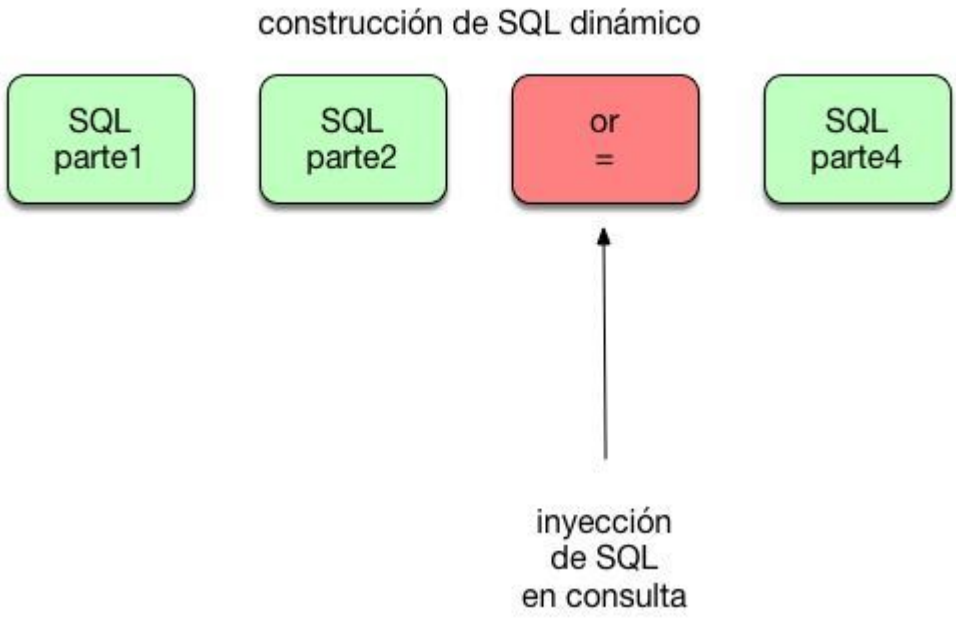
JPA SQL Injection

El programa principal se encarga de realizar una consulta JPA utilizando SQL dinámico y construyendo la consulta por partes.

En este caso estamos buscando una persona por nombre y apellidos , así pues pasamos ambos parámetros. El resultado por la consola confirma que la consulta funciona de forma correcta y que solo el registro de pedro cumple.

```
jul 03, 2017 9:40:55 AM org.hibernate.hql.internal.Q
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.nombre as nombre1_0_, pe
pedro
```

Ahora bien estamos ante una situación comprometida ya que estamos construyendo la consulta de forma dinámica y nos pueden realizar un ataque de SQL Injection. ¿Cómo nos pueden atacar ? . Muy sencillo es suficiente con modificar el parámetro que recibimos de apellidos por algo como “‘ or “=” “.



Esta modificación añadirá a la SQL una condición que siempre se cumple (inyectandola).

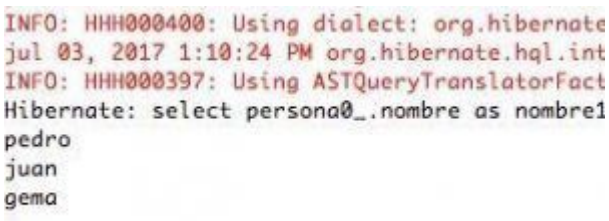
```
seleccionarPersona("pedro", <strong>"' or ''='");</strong>
```

La nueva SQL generada será:

select persona0_.nombre as nombre1_0_, persona0_.apellidos as apellido2_0_, persona0_.edad as edad3_0_ from Persona persona0_ where persona0_.nombre=’pedro’ and

persona0_.apellidos=” or ”=”

El resultado por pantalla cambia y nos muestra todos los usuarios de la tabla :



Tenemos un problema de JPA SQL Injection y hemos recibido un ataque. Para solventar estos casos deberemos utilizar el API de criteria que veremos en el próximo artículo.

Otros artículos relacionados:

1. Introducción a Spring Data y JPA
2. Un ejemplo de JPA Entity Graph
3. JPA Single Table Inheritance
4. JPA WikiPedia