# Python Programming

Módulo 4

# Prevención en MySQL

# Prevención en pymysql

En todos los motores de bases de datos, en todos los módulos para interactuar con ellos y en todos los lenguajes de programación, la lógica para prevenir una inyección de código SQL es la misma: no incluir variables cuyos valores provienen del usuario dentro de una consulta SQL sino dejar que el módulo se encargue de ello.

No obstante, cómo se realice esto último puede tener algunas variaciones respecto de lo que vimos anteriormente para SQLite.

Vamos a considerarlas en este apartado para el motor de base de datos MySQL usando el módulo PyMySQL (en Python existen otros módulos para acceder a bases de datos MySQL, como mysql-connector-python y MySQLdb).

Un programa análogo al anterior pero usando el módulo `pymysql` se vería más o menos así:

```
import pymysql

conn = pymysql.connect(
    host="localhost",
    user="usuario",
    passwd="clave",
    db="nombredb"
)
cursor = conn.cursor()

try:
   cursor.execute("CREATE TABLE personas (nombre VARCHAR(50), edad INT)")
except pymysql.err.InternalError:
   pass

nombre = input("Nombre: ")
edad = int(input("Edad: "))
cursor.execute(f"INSERT INTO personas VALUES ('{nombre}', {edad})")

conn.commit()
print("¡Datos ingresados correctamente!")

conn.close()
```

Según lo que hemos dicho, aquí tenemos el mismo problema con la siguiente línea:

```
cursor.execute(f"INSERT INTO personas VALUES
('{nombre}', {edad})")
```

No obstante, nuevamente, si ejecutamos el programa e ingresamos estos datos:

```
Nombre: Carlos', 30); DELETE FROM personas;
--
Edad: 30
```

Veremos que obtenemos la siguiente excepción:

```
Traceback (most recent call last):
    [...]
pymysql.err.ProgrammingError: (1064, "You have
an error in your SQL syntax; check the manual
that corresponds to your MySQL server version
for the right syntax to use near 'DELETE FROM
personas; -- ', 30)' at line 1")
```

Esto ocurre porque por defecto el módulo `PyMySQL` no permite ejecutar múltiples consultas en una misma llamada a `execute()`. Para hacerlo, debemos pasar el parámetro `client_flag=pymysql.constants.CLIENT.MULTI_STATEMENTS` a la función `connect()`.

```
conn = pymysql.connect(
    host="localhost",
    user="usuario",
    passwd="clave",
    db="nombredb",
    client_flag=pymysql.constants.CLIENT.MULTI
_STATEMENTS
)
```

Ahora repitamos los pasos y veremos que la consulta se ejecuta correctamente:

```
Nombre: Carlos', 30); DELETE FROM personas; --
Edad: 30
¡Datos ingresados correctamente!
```

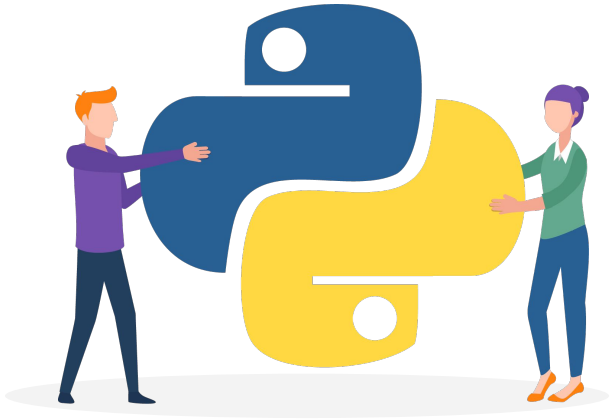
Por ende, la inyección de código SQL habrá sido satisfactoria y se habrán eliminado todas las filas de la tabla `personas`.

La solución para erradicar la vulnerabilidad es la misma que para el módulo `sqlite3`, pero, en este caso, usamos los caracteres `%s` en lugar del signo de interrogación.

```
cursor.execute("INSERT INTO personas VALUES (%s,
%s)", (nombre, edad))
```

No confundirse con el siguiente código, que sí genera la vulnerabilidad:

```
cursor.execute("INSERT INTO personas VALUES ('%s', %d)" % (nombre, edad))
```



# ¡Sigamos trabajando!