Python Programming

Módulo 4

Prevención en SQLite3

Prevención en SQLite3

La inyección de código SQL es un tipo de vulnerabilidad informática que permite a un usuario no autorizado ejecutar consultas SQL en una base de datos y que puede darse en cualquier tipo de aplicación (de consola, web, de escritorio, móvil). Para que un programa genere este tipo de vulnerabilidad es necesario que le permita al usuario ingresar algún dato y luego que este dato sea incluido dentro de una consulta a la base de datos.

Por ejemplo, consideremos el siguiente programa que solicita al usuario un nombre y una edad y luego inserta estos datos en una base de datos SQLite.

```
import sqlite3

conn = sqlite3.connect("personas.sqlite")
cursor = conn.cursor()

try:
   cursor.execute("CREATE TABLE personas (nombre TEXT, edad NUMERIC)")
except sqlite3.OperationalError:
   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()
```

En una ejecución típica de este código, la consola se vería más o menos así:

```
Nombre: Carlos
Edad: 30
¡Datos ingresados correctamente!
```

Ahora bien, los datos ingresados por el usuario son incorporados a la consulta SQL utilizando la sintaxis de Python de anteponer una `f` a la cadena y encerrar entre llaves los nombres de las variables cuyos valores queremos incluir. Más específicamente en la siguiente línea:

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

Si nombre es "`Carlos`" y edad es `30`, entonces la consulta que se ejecuta es la siguiente:

```
INSERT INTO personas VALUES ('Carlos', 30)
```

El problema con esto es que no se efectúa validación alguna de los datos ingresados por el usuario en la consola. ¿Qué ocurre si se ingresan los siguientes datos?

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

Según la lógica de nuestro programa, la consulta quedaría así (nótese que hay un espacio después de los dos guiones medios):

```
INSERT INTO personas VALUES ('Carlos', 30);
DELETE FROM personas; -- ', 30)
```

Puesto que en SQL se pueden separar varias consultas usando punto y coma, el código anterior realiza lo siguiente:

1. Insertar una fila con los valores “Carlos” y 30 como nombre y edad.
2. Eliminar todas las filas de la tabla.

El doble guión medio al final de la consulta introduce un comentario, es decir, los últimos caracteres `', 30)` son ignorados por completo, y esto es necesario para que el motor de bases de datos (en este caso, SQLite) no arroje ningún error de sintaxis. Entonces, hasta ahora, teóricamente habríamos logrado inyectar una consulta SQL (`DELETE FROM`) a partir de la función `input()`. Pero si ingresamos estos datos (es decir, el nombre y la edad tal como lo indicamos más arriba) veremos que el programa arroja el siguiente error.

```
Traceback (most recent call last):
    [...]
    cursor.execute(f"INSERT INTO personas
VALUES ('{nombre}', {edad})")
sqlite3.Warning: You can only execute one
statement at a time.
```

Nos encontramos con que el módulo SQLite, particularmente su función `execute()`, solo permite ejecutar una consulta a la vez, de modo que toda cadena que le pasemos como argumento y que posea más de una consulta (separadas por punto y coma) hará que el módulo `sqlite3` excepción anterior (nótese que esto ocurre con este módulo en particular y puede no aplicarse a otros módulos para interactuar con bases de datos).

Si queremos ejecutar más de una consulta con una misma cadena, debemos usar la función `executescript()`. Intentemos, entonces, reemplazar esta línea:

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

por esta:

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

Ahora ejecutemos nuevamente nuestro programa e ingresemos los siguientes valores:

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

Veremos que el programa no arroja ningún error y al abrir la base de datos nuestra tabla `personas` estará efectivamente vacía. ¡Hemos inyectado correctamente código SQL!

Pues bien, ¿cómo hacemos para prevenir esta vulnerabilidad? Recordemos que su origen radica en incorporar variables cuyos valores vienen del usuario dentro de la consulta SQL. El método por el cual se efectúe esta operación es indistinto, es decir, puede ser cualquiera de los siguientes:

- `.execute(f"INSERT INTO personas VALUES ('{nombre}', {edad})")`
- `.execute("INSERT INTO personas VALUES ('{}', {})".format(nombre, edad))`
- `.execute("INSERT INTO personas VALUES ('" + nombre + "', " + str(edad) + ")")`
- `.execute("INSERT INTO personas VALUES ('%s', %d)" % (nombre, edad))`

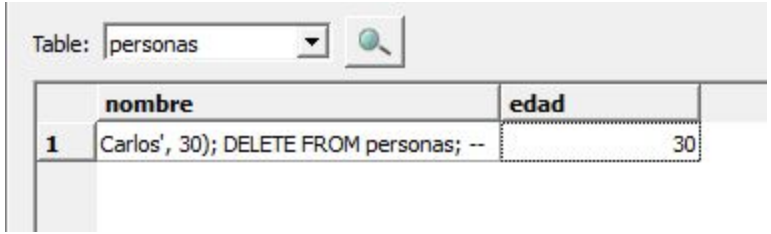
Estos cuatro códigos generan la vulnerabilidad de inyección de SQL. La única forma de prevenirla es chequear que los valores de `nombre` y `edad` (o cualquier otra variable que sea incluida dentro de la consulta) no contengan código SQL. No es necesario que hagamos este trabajo manualmente, la función `execute()` lo hará por nosotros si pasamos esos datos en una tupla como segundo argumento y, en la consulta, utilizamos un signo de interrogación para indicar los lugares en donde deben ubicarse dichos datos.

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

Nótese que, por más que `nombre` sea una cadena, dentro de la consulta no usamos comillas simples alrededor del signo de interrogación. Ahora, ejecutemos nuevamente el programa con esta línea actualizada e ingresemos los datos para inyectar código:

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

En efecto los datos se ingresaron correctamente, pero no hubo inyección de código alguna. Esto es, la consulta `DELETE FROM personas` no se ejecutó. Más bien, el resultado es el siguiente:



Lo cual es totalmente lógico, puesto que hemos dicho que el nombre que queríamos ingresar era `Carlos', 30); DELETE FROM personas; --` (un nombre un poco raro). Así hemos eliminado la posibilidad de inyección de código SQL en nuestra aplicación.

¡Sigamos trabajando!