Route/URL (Scan 1)	Parameter	Number of Injection Trials	Number of Successful Trials
http://127.0.0.1:8081/login	email	14	0
http://127.0.0.1:8081/login	password	15	0
http://127.0.0.1:8081/register	email	14	0
http://127.0.0.1:8081/register	name	15	0
http://127.0.0.1:8081/register	password	7	0
http://127.0.0.1:8081/register	password2	15	0
Route/URL (Scan 2)	Parameter	Number of Injection Trials	Number of Successful Trials
http://127.0.0.1:8081/login	email	14	0
http://127.0.0.1:8081/login	password	15	0
http://127.0.0.1:8081/register	email	14	0
http://127.0.0.1:8081/register	name	15	0
http://127.0.0.1:8081/register	password	15	0
http://127.0.0.1:8081/register	Password2	15	0

1. All the user input fields that are accessible from the home page which include the email text field and password text field are covered. The second scan with the session ID should be able to access the fields in update profile (Name, Shipping Address, Postal Code) and create product (Price, Title, Description) however it does not. There are no successful exploits.
2. The results are different due to the addition of the session ID in the second round of scanning. This session ID corresponds to a verified registered user with a specific username and password attached. The purpose of adding in the session ID is to allow the SQL injector to log into a user that the session ID corresponds to which allows the SQL injector to scan restricted forms that are only available after login which for our project includes Update Profile and Create Product and their corresponding fields (Name, Shipping Address, Postal Code) in Update Profile and (Price, Title, Description) in Create Product, these are fields that would not be accessible for injection code without a session ID.
3. The injection payload hacks into the database and alters the data without being stopped by the protection in the service layer. This is because the injection payload exists on the application level and is interpreted by the database which will run anything the payload asks so long as the injection goes through. This injection payload is to fingerprints the database by checking on each of the user inputs, in the case of the first scan this includes the inputs for email, name, password, and password2. Since the user form does not display the result of the inputs MySQL can automatically fire out queries to guess towards the information in the database and look for any exploitable attributes which is the purpose of this injection payload. The error message is not always accessible to test how the program reacts to a query which is why this injection payload also uses timing attacks which are seen when the tests use the sleep command. The most standard SQL injection is 1st order injection but that is done when we can return the results from the malicious input.