2 - SQL Injection and ORMs

1. Learning Outcomes

On completion of this lab you will have:

- Demonstrated how SQL injection can be performed on a badly implemented RDMBS back-end interface
- Implement SQL-injection proofing in your implementation
- Implemented an API model layer using the Sequelize object relational mapper
- Implemented API in Express using an ORM-based model layer

2. Organisation

Please complete the exercises individually.

3. Grading

This worksheet is worth up to 7.5% of your overall module grade. You must attend and sign in at 10 labs in order to obtain full credit for your submitted worksheets. You may work on this worksheet during lab 3 and lab 4 with instructor assistance. You must also demonstrate your submission in order to receive credit - see below.

4. Submission

The deadline for submission is Sunday Feb 26, 2017 @23:59 through Webcourses.

5. Demonstration

You will give a brief demonstration of your submission to the lab instructor in lab 5.

6. Requirements

For this lab you will need to

- Use your own laptop with local tools or,
- Sign up for a free account with a cloud provider such as Nitrous.IO and Heroku

7. Resources

You are free to research whatever you need to solve the problems in this lab. Some recommended resources include:

- https://nodejs.org/en/
- http://expressjs.com/
- http://www.unixwiz.net/techtips/sql-injection.html
- http://mherman.org/blog/2015/10/22/node-postgres-sequelize/#.WJ9-aBKLSRt
- http://massive-js.readthedocs.io/en/latest/
- http://docs.sequelizejs.com/en/v3/

8. Problem Sets

The following platform-independent tasks can be solved on Windows, Mac local Linux or Cloud Linux as you prefer

For your lab submission, take screenshots or cut-n-paste your solutions into a document which you will submit through webcourses. Include your code or provide a link to a code repository where your code lives

1	Building on your solution to worksheet 1 for the API to the `products` resource from the `pgguide` database, extend the product indexing endpoint to allow the filtering of products by name as follows `GET /products[?name=string]` For your solution you should implement the query in such a way as to allow an attacker to inject arbitrary SQL code into the query execution. Show, using your badly implemented approach, how an attacker can craft a query string to allow the dropping of the products table. For convenience, you can continue to use MassiveJS to interface with the database.	15 Marks
2	Provide two solutions to eliminate the security hole in your approach from section 1 as follows: ● Using a parameterised query ● Using a stored procedure using SQL or PLPGSQL whichever you prefer Explicitly show that the attack from section 1 is not now possible for each of your solutions Again, you can just use MassiveJS as your database interface library here too.	15 Marks

3	Create a brand new Express project using the Sequelize ORM. Install and configure Sequelize and wire it up to a new PostgreSQL database, including the connection string, username and password. Verify that you have basic connectivity before proceeding. Next, you will build a simple scheme for a hypothetical civil courts of justice system. Create migrations for the following models and (attributes): ● `Judge (id:serial, name:string, room:number, ext:string)` ● `Courtroom (id:serial, number:string)` ● `Participant (id:serial, name:string, address:string, type:string) where type is either “claimant” or “respondent”` ● `Case: (judge_id:integer, courtroom_id:integer, claimant_id:integer, respondent_id:integer, start_date:date, duration:interval, result:boolean) where a true result means that the claimant wins, false if loss and null if not known` Ensure that the appropriate associations and referential integrity checking are set up in your models	25 Marks
4	Use your models and Javascript code to populate the database with test data for all of the models above	10 Marks
5	Build a full RESTful API using Sequelize and Express for the courts system. Your API should support the CRUD operations of each of the models returning JSON responses Show test cases for each of the API endpoint REST operations	25 Marks
6	Add validating logic in your solution to ensure that a courtroom cannot be double-booked when creating a new case	10 Marks