Security Practical 3

Dr Chris G. Willcocks

Email: christopher.g.willcocks@durham.ac.uk

Practical 3

Background

Welcome to the third practical. In this practical we will be implementing what we learnt in the lecture on databases and extending our web server accordingly. We will be using a small, fast, and lightweight SQL database called 'sqlite' primarily for its simplicity; however I recommend you also check out NoSQL databases such as MongoDB for other applications. We will be doing injection and inference attacks. This will also be a good opportunity for those already familiar with SQL to get some additional practice.

Tasks

1. It is important that you are familiar with the SQL syntax in order to understand some of the more obscure queries/inference attacks.

   (a) Complete the SQL tutorial at: `https://www.w3schools.com/sql/sql_intro.asp` (you have to click 'Next' at each stage).

   (b) At the end of the tutorial you will be given a SQL quiz. Complete this. You are expected to score above 20 on this quiz.

2. Change directory to '5' then run `server.py`:

   (a) The following line, in `server.py` will look to see if there is a database file, if it is not there it will create it:

   Listing 1: Python

   ```
   Handler.conn = sqlite3.connect('secret.db') # connection to database
   ```

   (b) You should now see a new database file 'secret.db' has been created.

   (c) Open this file in sqlitebrowser (`https://sqlitebrowser.org/`), which is an open source database graphical browser for sqlite. Windows MDS users can launch this from the AppHub.

   (d) Click the 'Execute SQL' tab, and type your SQL commands here to edit/query the database (you can also do this directly from python/ipython, but its easier with the DB Browser GUI).

   (e) **Save a backup history of all your SQL statements in the following sections.**

   (f) Create a 'Users' table with the following data, where the ID field is the primary key and id and username have the NOT NULL constraint, whereas the other columns can accept NULL values. Grades and fines are floating point numbers. Make all text NVARCHAR(80) for now.

ID	Username	Firstname	Lastname	Grade	College	Fines
1	jess	Jess	Adams	86.4	Castle	4.21
2	greg7	Greg	Courier	62.0	Chads	0.0
3	alice4	Alice	Smith	57.9	Castle	1.21
4	chrzi	Chris	Jackson	73.8	Cuths	5.33
5	lauran3	Laura	Walker	21.2	Ustinov	0.34
6	ai219	Andrea	Ivanov	84.2	Ustinov	0.92
7	seb123	Seb	Elbert	17.3	Chads	0.0

With the table now created, you can paste the following for your insert statements:

Listing 2: SQL

```
INSERT INTO Users VALUES(1,'jess', 'Jess', 'Adams', '86.4', 'Castle', 4.21);
INSERT INTO Users VALUES(2,'greg7', 'Greg', 'Courier', '62', 'Chads', 0);
INSERT INTO Users VALUES(3,'alice4', 'Alice', 'Smith', '57.9', 'Castle', 1.21);
INSERT INTO Users VALUES(4,'chrzi', 'Chris', 'Jackson', '73.8', 'Cuths', 5.33);
INSERT INTO Users VALUES(5,'lauran3', 'Laura', 'Walker', '21.2','Ustinov', 0.34);
INSERT INTO Users VALUES(6,'ai219', 'Andrea', 'Ivanov', '84.2', 'Ustinov', 0.92);
INSERT INTO Users VALUES(7,'seb123', 'Seb', 'Elbert', '17.3', 'Chads', 0);
SELECT * FROM Users;
```

(g) Also, create a 'Private' table with the following data, where ID is a primary key and UserID is a foreign key. Store the wallets as NVARCHAR(80):

ID	UserID	Wallet
1	3	KworuAjAtnxPhZARLzAadg9WTVKjY4kckS8pw38JrD33CeVYUuDm
2	1	Kwi5LPxVehUieD18AXiXTay9UDkRC7wLShe4tR5kzym1k2NhzEQ6
3	5	L4mGG15YacXWPU7LHM8Lj2LboxabRriZGHFZb5eLDN7mPXPPAHQF

You can use the following for your insert statements:

Listing 3: SQL

```
INSERT INTO Private VALUES(1, 3, 'KworuAjAtnxPhZARLzAadg9WTVKjY4kckS8pw38JrD33CeVYUuDm');
INSERT INTO Private VALUES(2, 1, 'Kwi5LPxVehUieD18AXiXTay9UDkRC7wLShe4tR5kzym1k2NhzEQ6');
INSERT INTO Private VALUES(3, 5, 'L4mGG15YacXWPU7LHM8Lj2LboxabRriZGHFZb5eLDN7mPXPPAHQF');
```

(h) Do an inner join on the 'Users' and 'Private' table data, such that the query result looks like:

Firstname	Lastname	Wallet
Alice	Smith	KworuAjAtnxPhZARLzAadg9WTVKjY4kckS8pw38JrD33CeVYUuDm
Jess	Adams	Kwi5LPxVehUieD18AXiXTay9UDkRC7wLShe4tR5kzym1k2NhzEQ6
Laura	Walker	L4mGG15YacXWPU7LHM8Lj2LboxabRriZGHFZb5eLDN7mPXPPAHQF

(i) Check that all the data in the database is correct (use the 'Browse Data' tab).

(j) Save your statements for creating the tables somewhere for later.

(k) Click 'Close Database' and save the changes (write them to 'secret.db').

3. SQL injection attacks:

   (a) Open the website and do an SQL injection attack on the username input field to get all the 'Users' data.

      i. Remember '−−' is a comment.

      ii. You may wish to study the source code of `server.py`

4. Prepared statements (parameterized queries):

   (a) Fix the server from SQL injection attacks by using parameterized queries.

      i. sqlite3 specific guide: `https://docs.python.org/2/library/sqlite3.html#sqlite3.Cursor.execute`

   (b) Try to do an SQL injection attack again on the username field to confirm that your prepared statements work as expected. Also confirm that you can still 'login' as normal users.

5. Inference attacks:

   (a) The instructor decides it would be helpful to put some class statistics on the website. The university has a policy in place where students should not be able to view the grades of other students. In another part of the website the instructor has a list of students by their college.

   (b) Can you infer the grades of each student based on the two views of the data?

      i. Just by using the website, what are the grades of Alice, Laura, and Seb?

(c) The instructor still wants to display a list of anonymous grades for each student, but has asked you to make the list anonymous so you can't work out which grade corresponds to which student. Edit `server.py` and alter the SQL queries to make the list of grades anonymous.

(d) Write SQL queries in `server.py` to return the average class grade and fines by college accordingly.

i. Confirm the website returns the following:

A. Average class grade:

57.542857142857144

B. Fines by college:

Castle	5.42
Chads	0.0
Cuths	5.33
Ustinov	1.26

(e) The university has a policy in place where students should not be able to view the fines of other students.

i. Login as Alice (her username is alice4).

ii. What are the fines of Greg, Seb, Chris, and Jess?

(f) What are your options for protecting against this?

6. This concludes the third practical, discuss and test your fully mitigated server with the person next or demonstrator and see if you can add any further security improvements. If you finish early, you may wish to extend the server to handle authenticated login with hashed and salted passwords stored in the SQL database.
