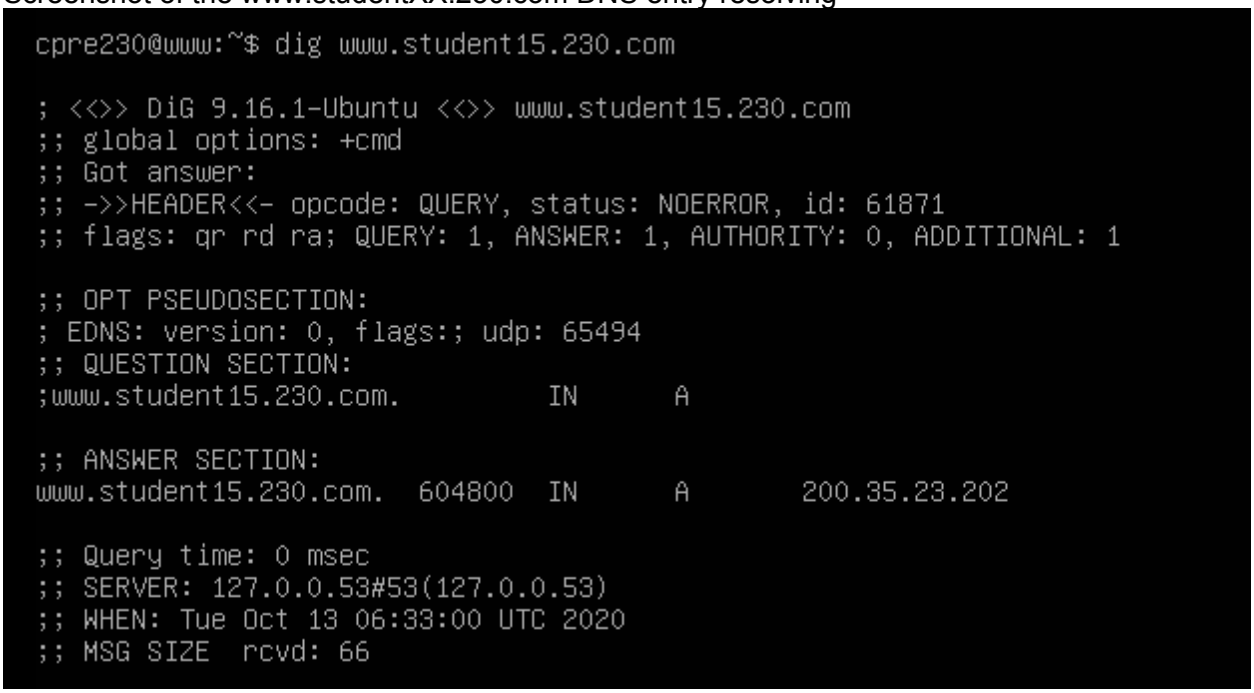
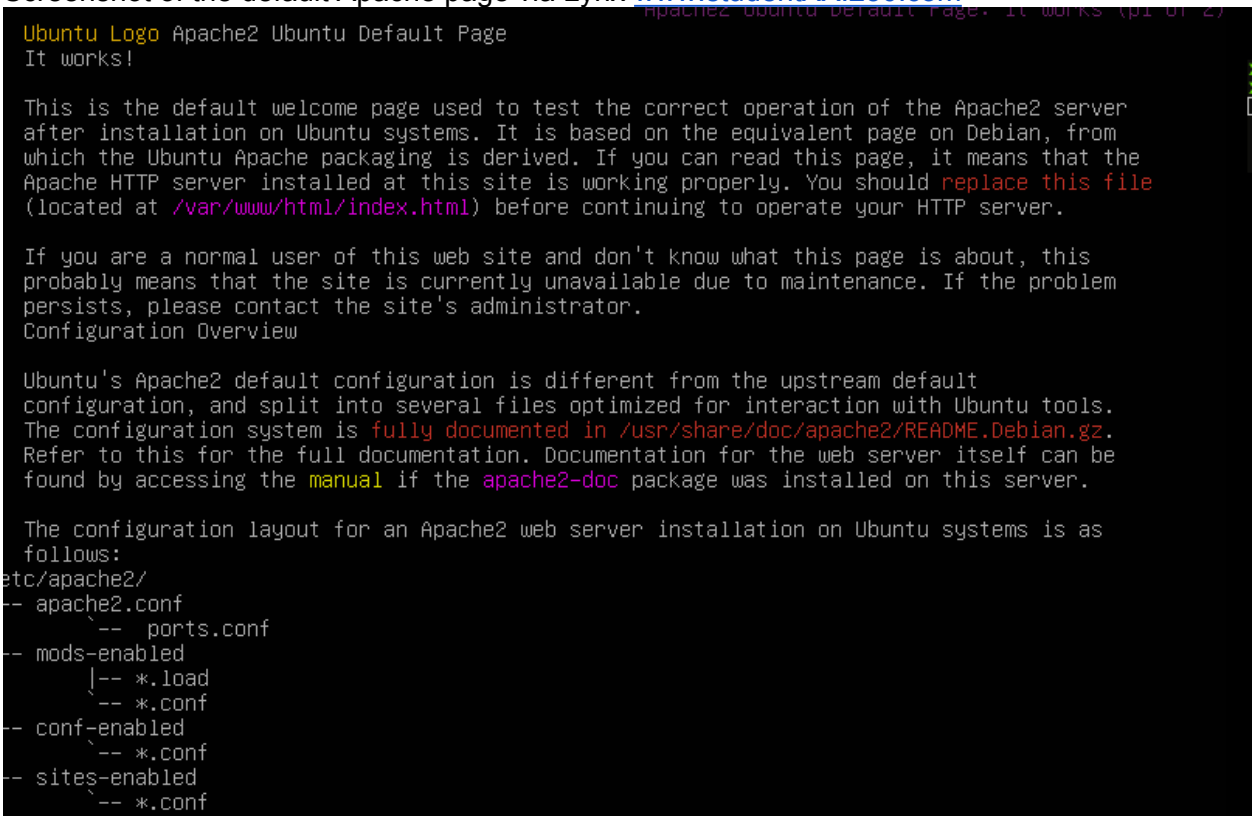
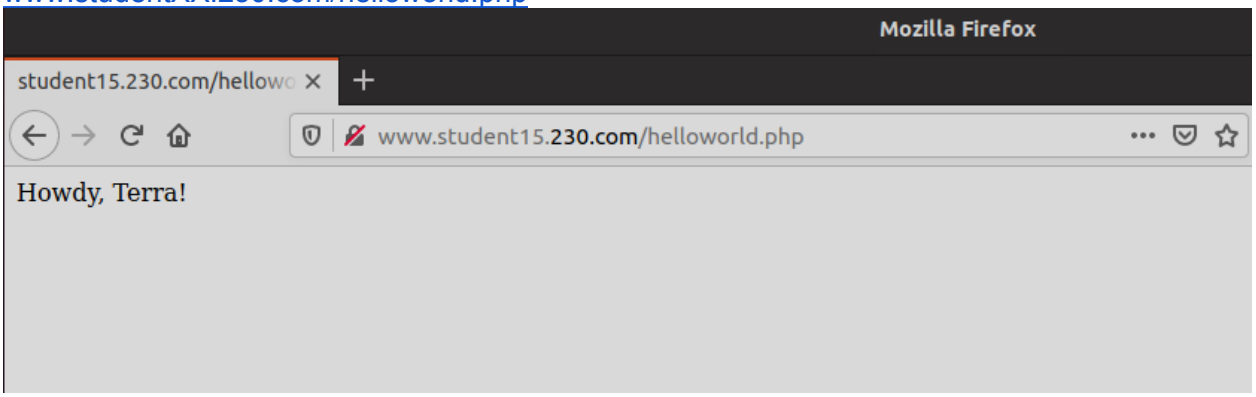
1. Screenshot of the www.studentXX.230.com DNS entry resolving



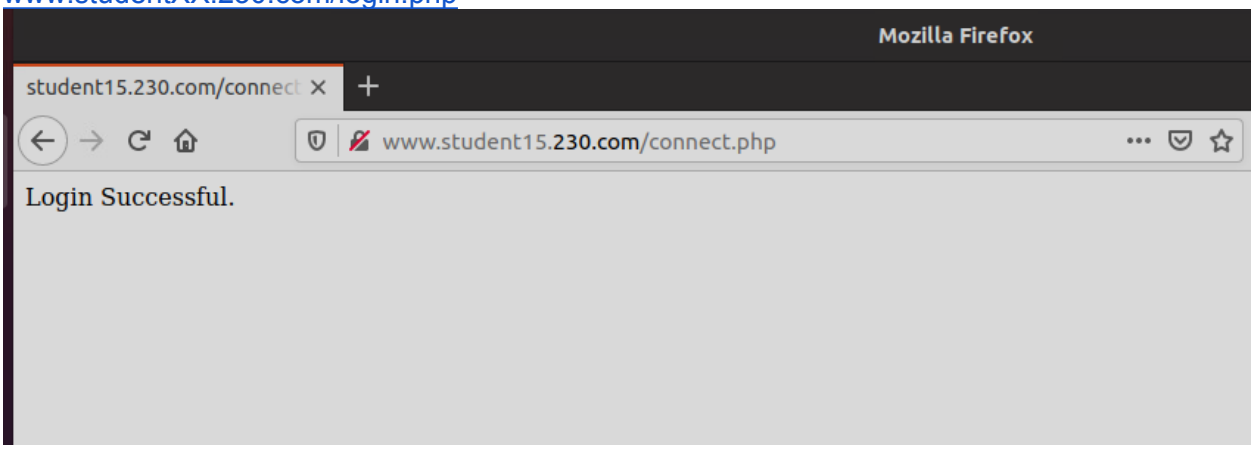
2. Screenshot of the default Apache page via Lynx www.studentXX.230.com



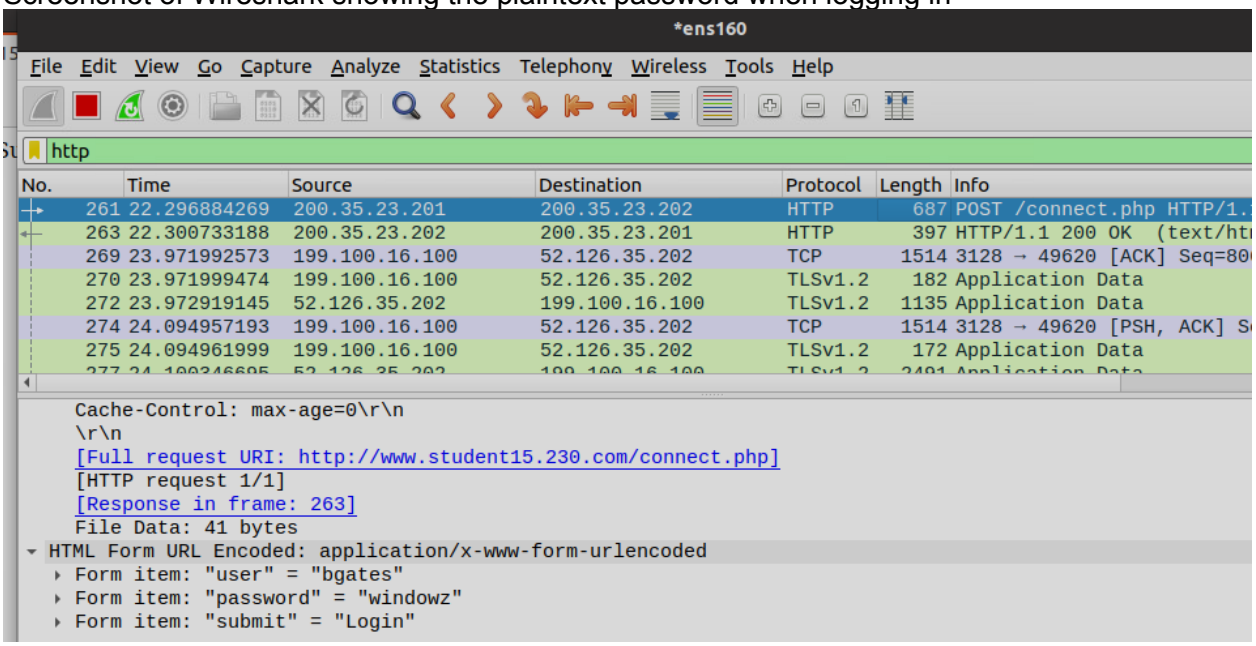
3. Screenshot of the Hello, world! PHP page via Firefox
www.studentXX.230.com/helloworld.php



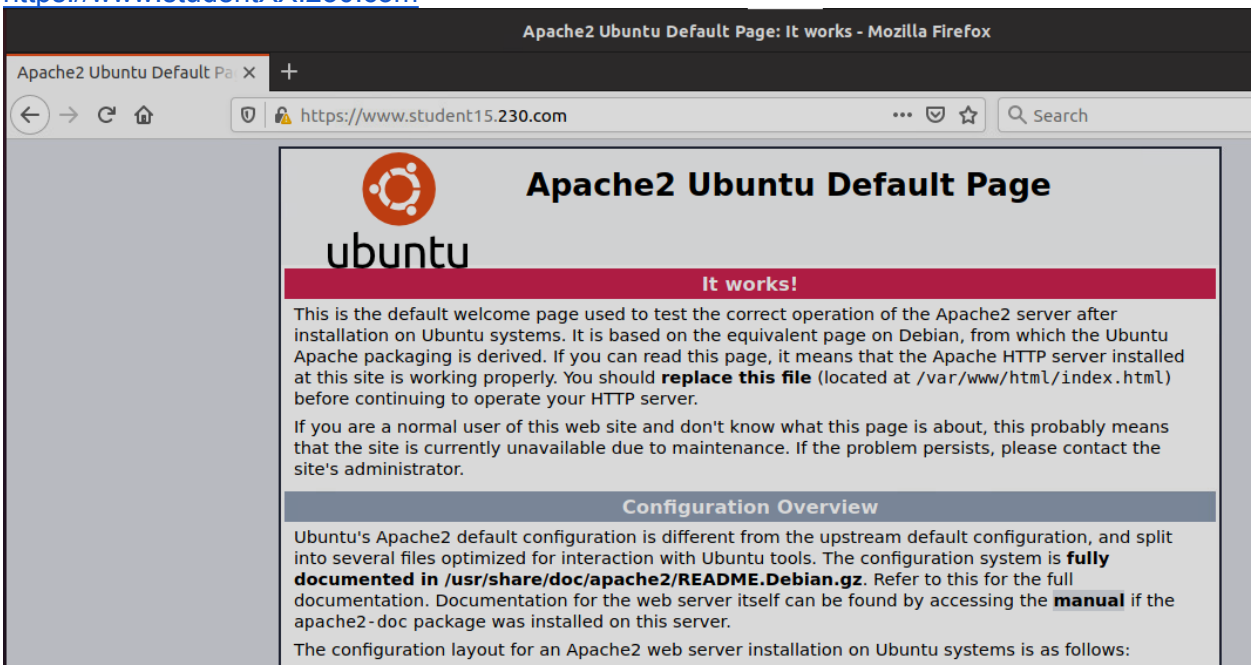
4. Screenshot of one of the users logged into PHP web page via Firefox
www.studentXX.230.com/login.php



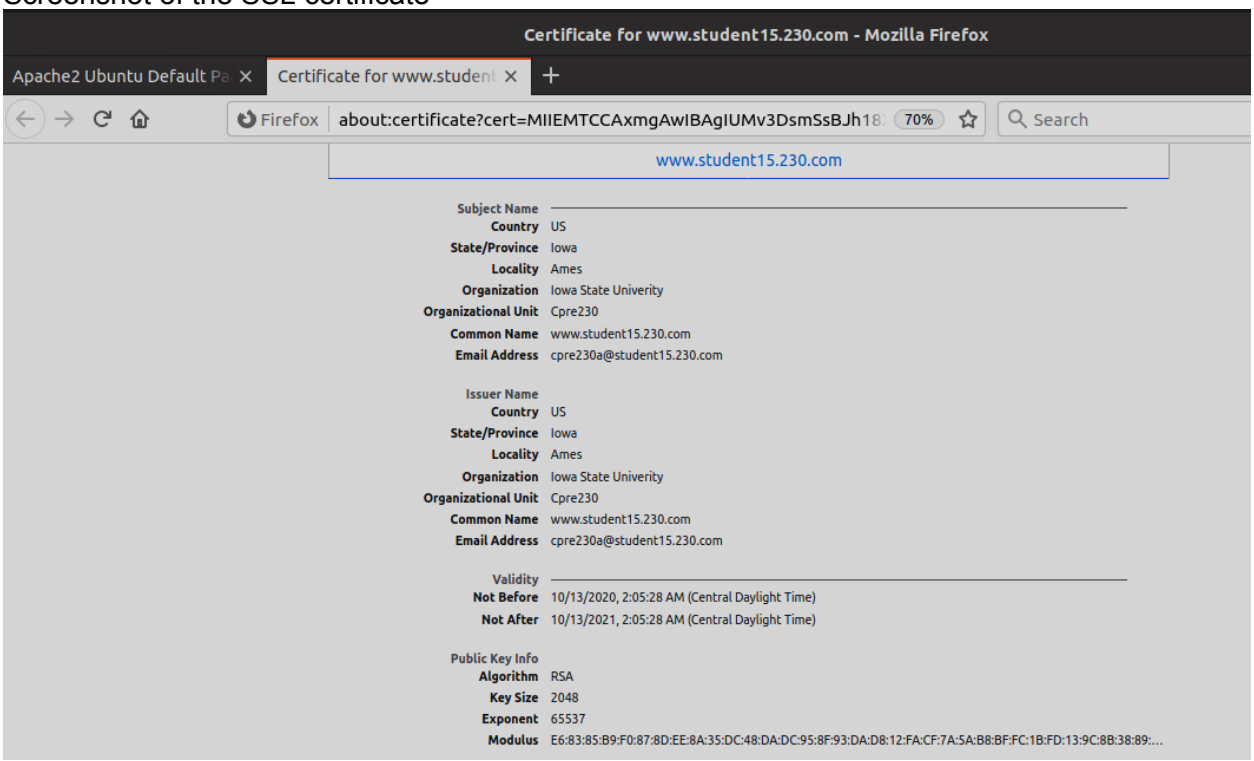
5. Screenshot of Wireshark showing the plaintext password when logging in



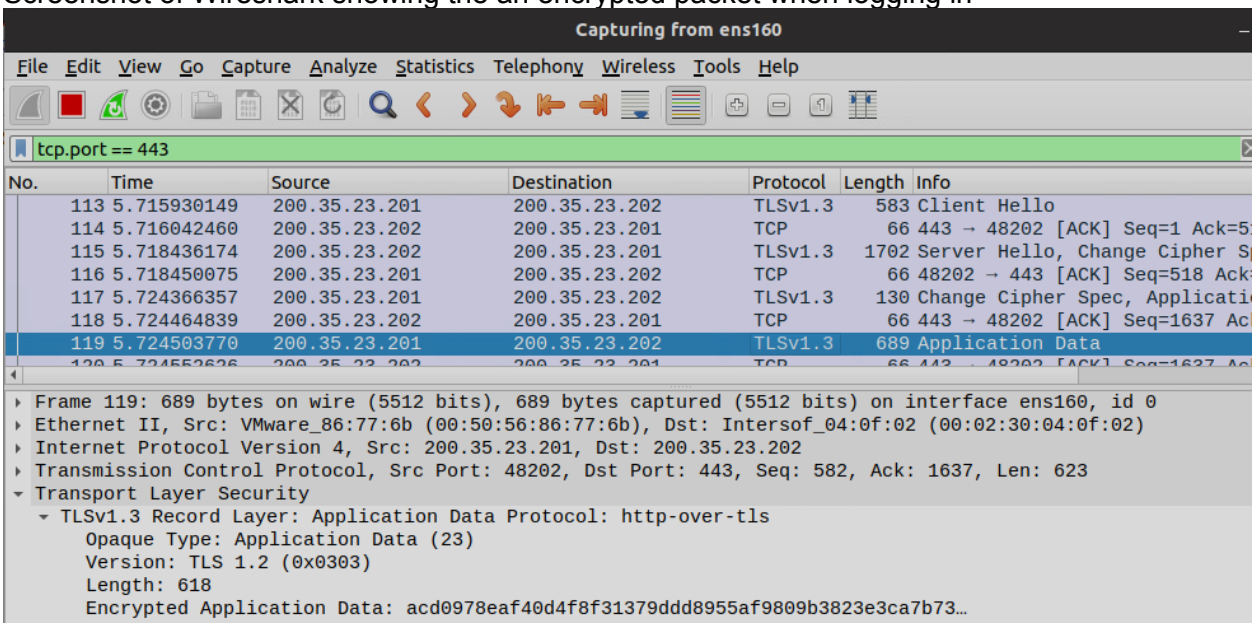
6. Screenshot of the Apache default page on HTTPS via Firefox: https://www.studentXX.230.com



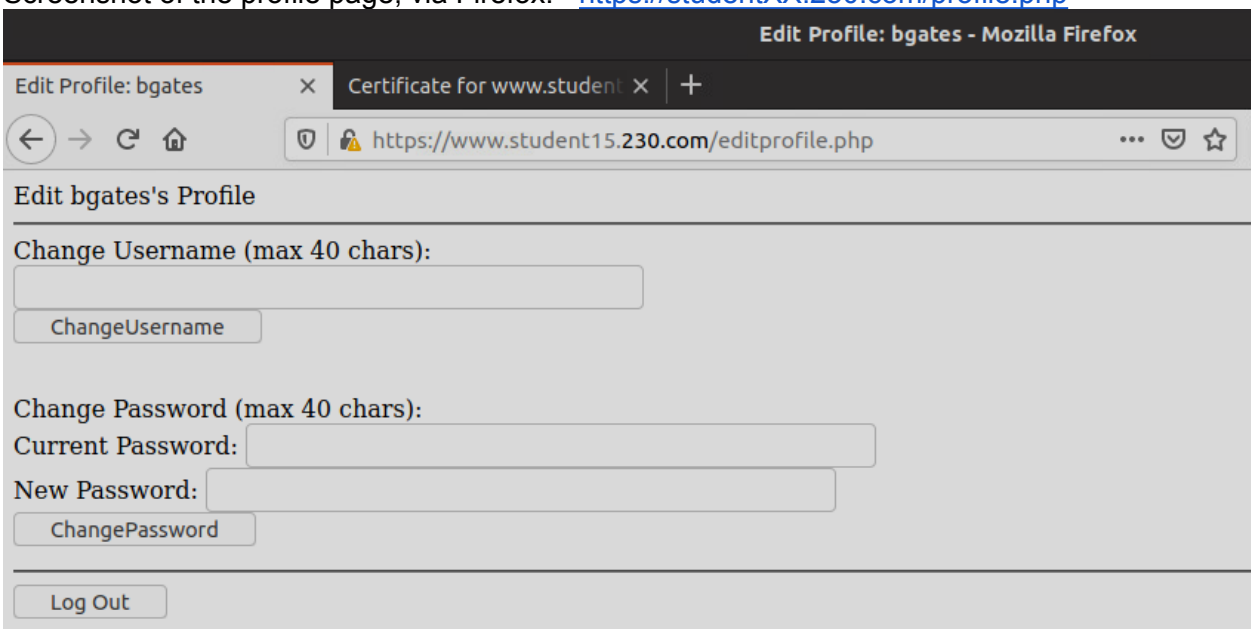
7. Screenshot of the SSL certificate



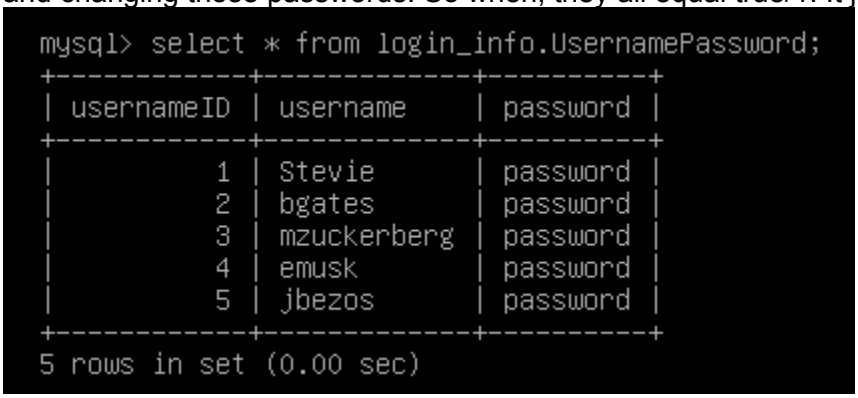
8. Screenshot of Wireshark showing the an encrypted packet when logging in



9. Screenshot of the profile page, via Firefox: https://studentXX.230.com/profile.php



10. Screenshot of the database, with changed username and password
    (I entered the ‘ OR ‘1’ = ‘1 in the current password field, and it set all the passwords to “password”. It makes sense though. It's probably just checking which passwords match and changing those passwords. So when, they all equal true/1. It just sets them all.)



```
mysql> select * from login_info.UsernamePassword;
+------------+-------------+----------+
| usernameID | username    | password |
+------------+-------------+----------+
|          1 | Stevie      | password |
|          2 | bgates      | password |
|          3 | mzuckerberg | password |
|          4 | emusk       | password |
|          5 | jbezos      | password |
+------------+-------------+----------+
5 rows in set (0.00 sec)
```

11. Description of what SQL injection is, and how you might go about preventing it

It seems like SQL injection is basically passing SQL code into a web interface like a text box or password box. This text gets passed to the server and handled by some set of sql code, so the passed code may be executed to some effect. This then can set values in SQL or retrieve passwords or ...

To prevent it, it would seem like you could check for SQL commands / statements within your text before it is handled by SQL and deny the use of that text. This, if done perfectly / without flaws, would completely prevent the use. Another idea may be preventing the use of certain characters that are commonly used/needed in SQL injections. As well, you could minimize its potential threat with a principle of least access, so, if I did what I did in the picture above, I wouldn’t be able to change let’s Stevie and bgates. Since those two are admin users and can’t be changed through the web interface at all.