**Internet Security – COS80013 Lab - 9 Report**

Internet Security – COS80013 Lab - 9 Report  
Student ID: 104837257  
Student Name: Arun Ragavendhar Arunachalam Palaniyappan  
Lab Name: COS80013 Lab 9 – SQL injection

Date: 16 /05/2025  
Tutor: Yasas Akurudda Liyanage Don

**Title and Introduction**

This lab was about understanding how SQL injection works and how attackers can use it to bypass logins or extract data. The setup included two virtual machines—Windows XP and RedHat Linux. From the XP machine, different SQL injection inputs were tested on a login page. The attacks worked because input checks were turned off on the server. The aim was to see how these flaws occur, what they allow, and how they can be stopped.

**Methodology**

The lab began by launching both the Windows XP and RedHat Linux virtual machines. On the XP browser, safelogin.php was opened and a normal login was done using "Warren" and "Eclipse". It worked as expected. After logging out, login.php was visited. This version had security checks and input sanitisations disabled.

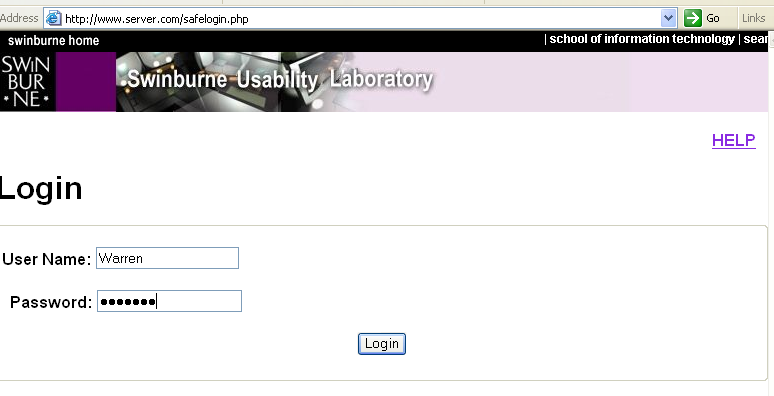
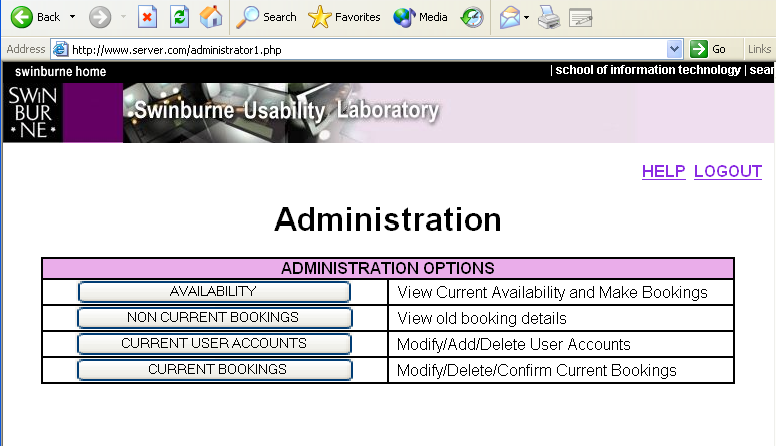
The same login failed here because the password wasn’t hashed. Then, an SQL injection was tested using x' or 'x' = 'x as the password, which bypassed the login. This worked because it tricked the SQL query into always returning true.

Other common injections were tried. The string ' or 1=1;-- didn’t work due to extra filtering. Column guessing was tested using x' and user = 'x, which showed an error, while x' and username = 'x did not—confirming username was valid. Table guessing was done using dot notation like users.username, which worked, while ACCOUNT.username failed.

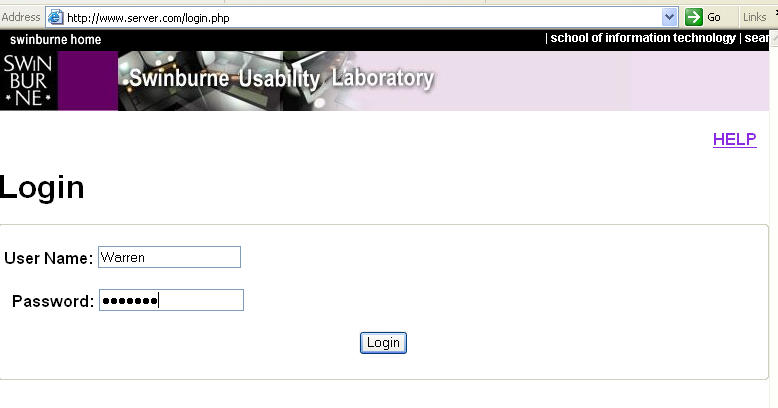
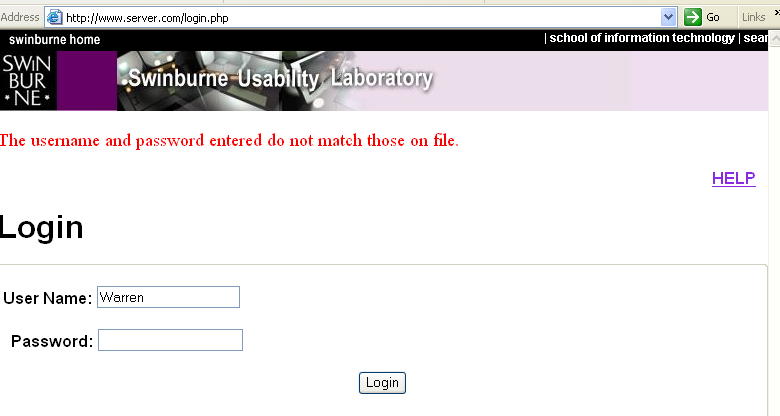
A LIKE query using x' or username LIKE '%admin% showed that an admin-style account existed. Inputs like x' or password = '' and x' or password IS NULL-- were used to check for accounts without passwords. Advanced attack strings (like insert, update, drop) were noted but not executed, as the server wasn’t set up for that level of injection.

**Data Recording and Screenshots**

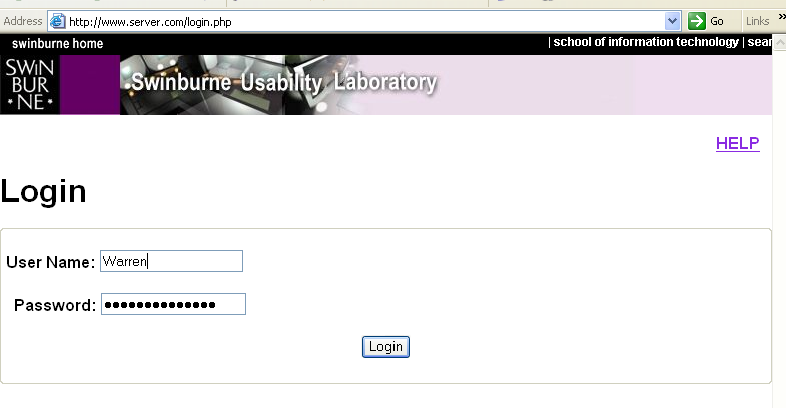
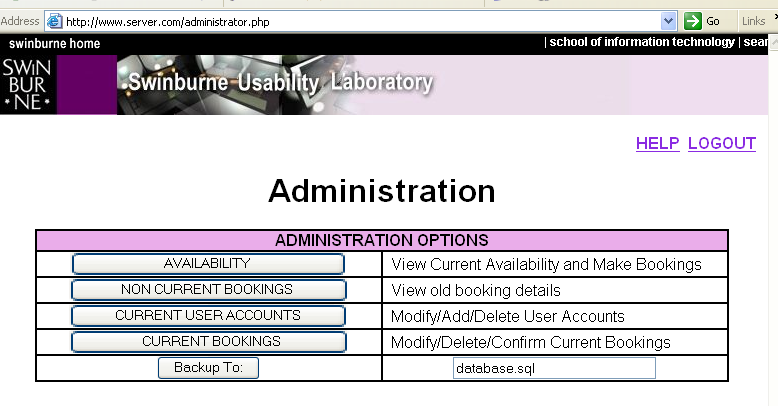
A normal login using safelogin.php with “Warren” and “Eclipse” worked correctly.

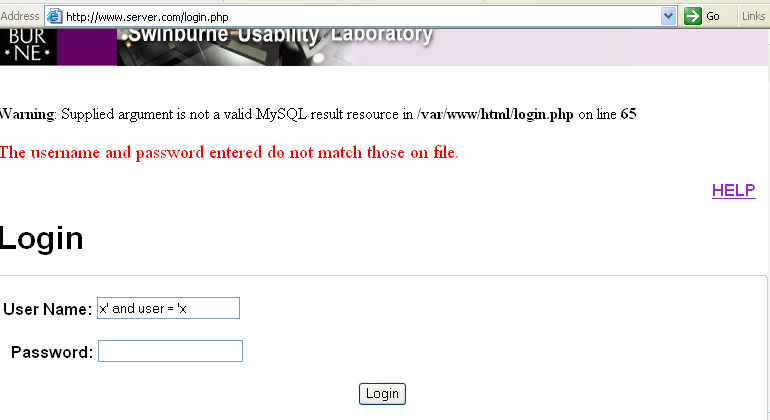
The same login failed in login.php due to hashing differences.

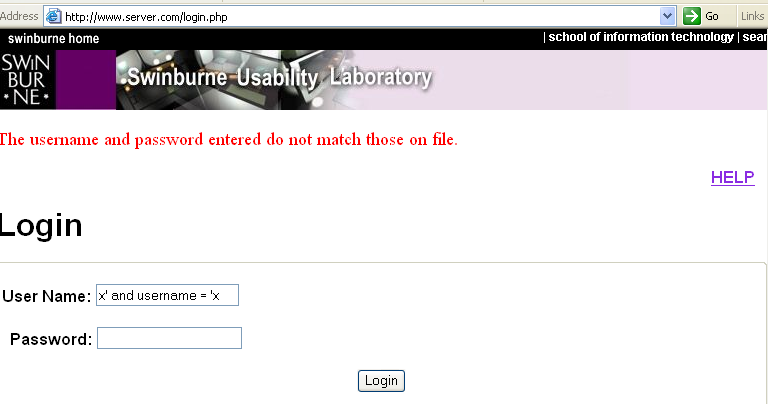
The input x' or 'x' = 'x successfully bypassed the login in login.php.

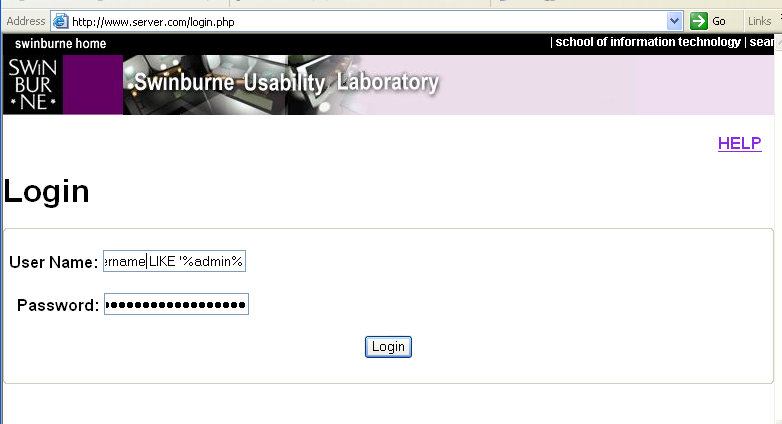
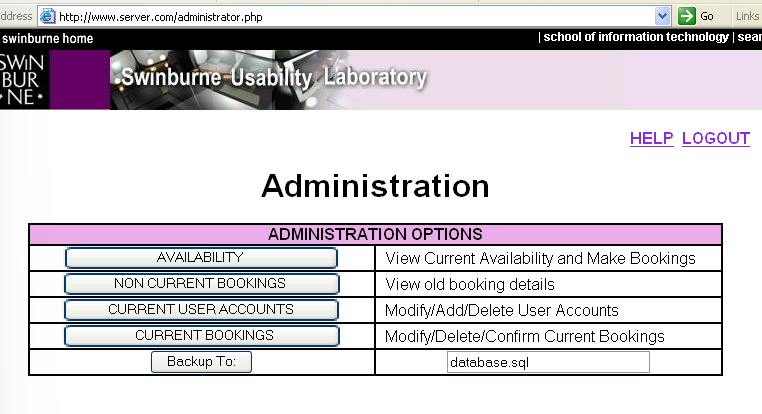
Trying x' and user = 'x showed a database error.

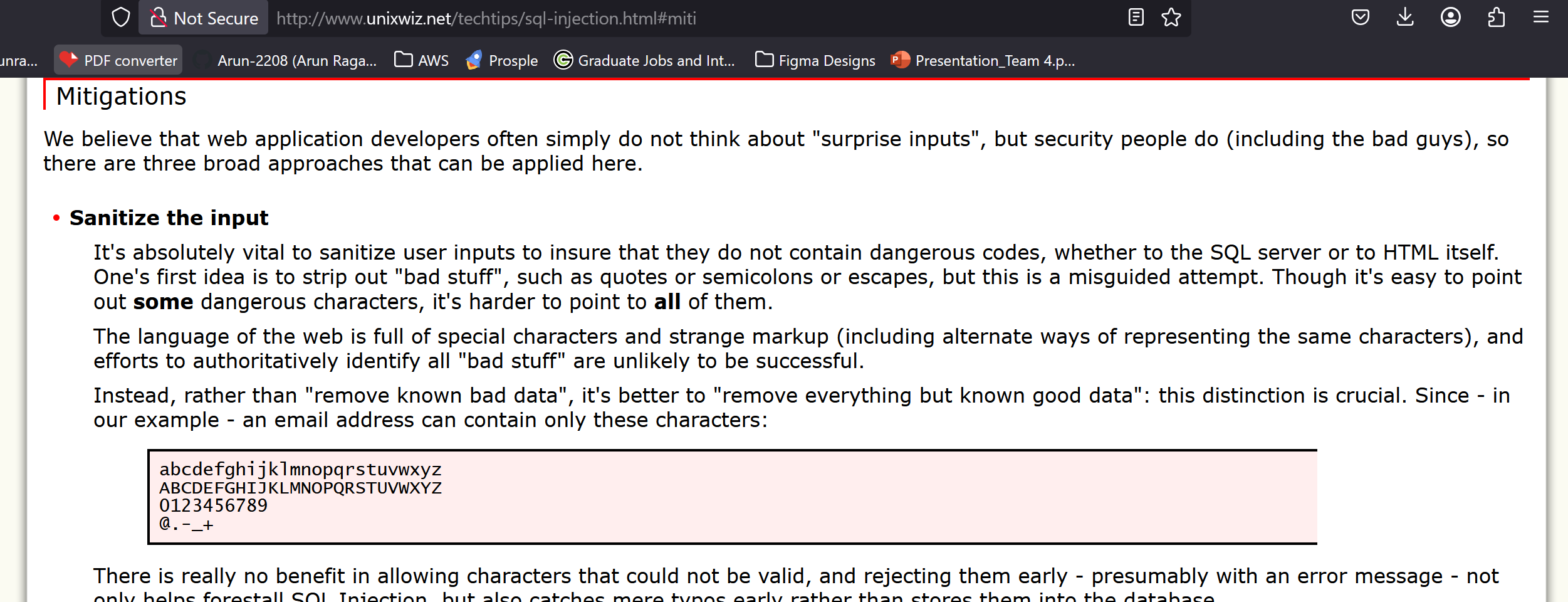
Trying x' and username = 'x did not show a database error, confirming that the column existed.

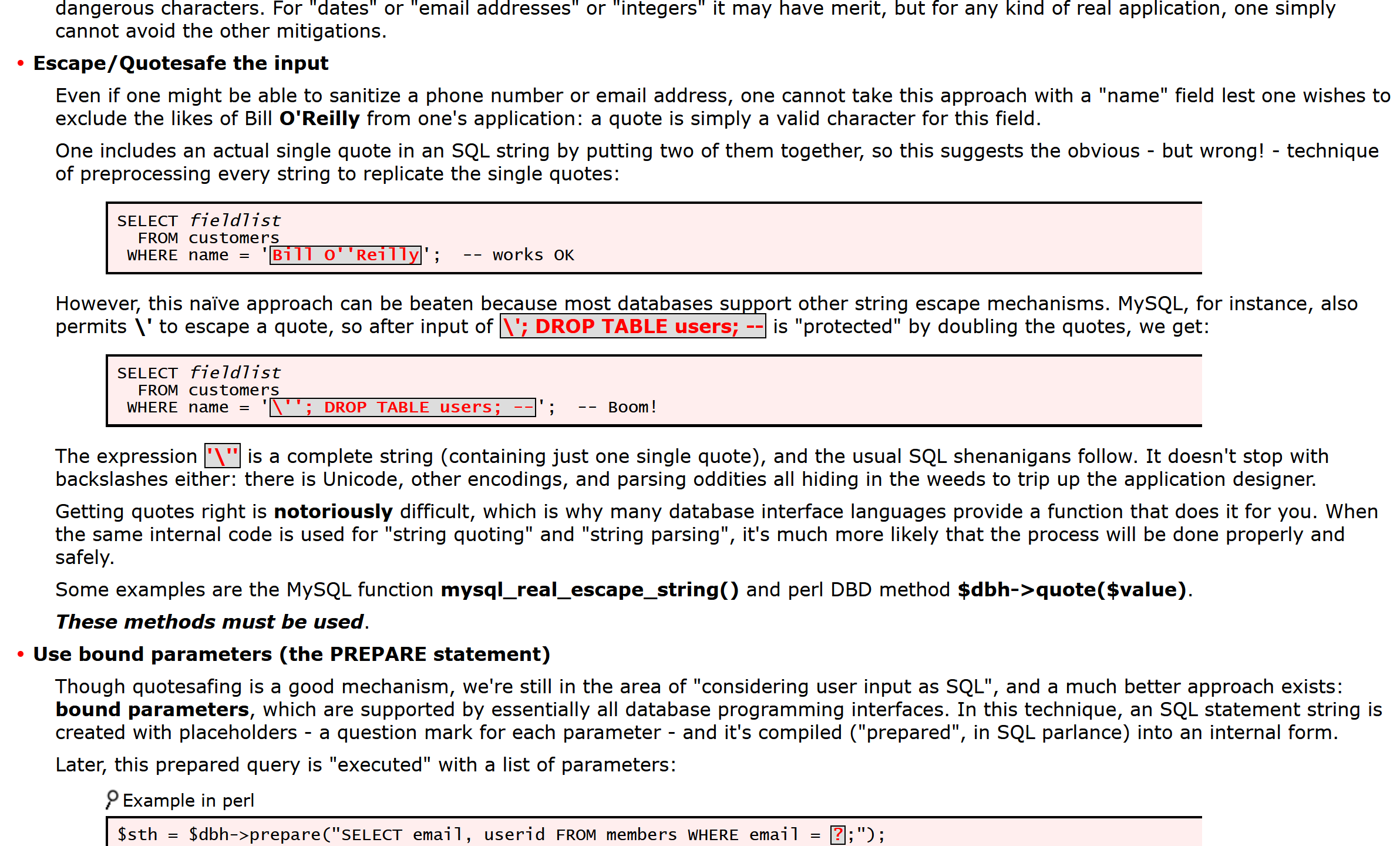
 

Using x' or username LIKE '%admin% revealed that an admin-like user existed.

Studied about the 7 techniques to prevent SQL injection attacks from <http://www.unixwiz.net/techtips/sql-injection.html>





**Discussion and Application of Learnings**

**Learning 1**  
The string x' or 'x' = 'x was able to log in without knowing the password. It worked because the SQL query was changed to always return true.  
**Real-World Application**  
If a website does not properly clean input fields, attackers can bypass logins and gain access as any user.

**Learning 2**  
Using made-up column names like user caused an error, but using real ones like username did not. This showed that attackers can guess a system’s database structure.  
**Real-World Application**  
By watching how the system responds to errors, attackers can slowly map out database columns and tables.

**Learning 3**  
The input using LIKE helped find usernames that contain certain words. This makes it possible to identify accounts like “admin” without needing a full username.  
**Real-World Application**  
This method can help attackers find target accounts or test for weak usernames.

**Learning 4**  
Testing for empty passwords showed that accounts with no password could be exploited.  
**Real-World Application**  
Some systems don’t force password creation, so attackers can find and take over these accounts.

**Learning 5**  
Some SQL commands like insert, update, or drop were not run here, but they show how much damage is possible if input is not filtered.  
**Real-World Application**  
In real attacks, these kinds of commands can change data, create fake users, or even delete entire tables.

**Limitations**

The lab used a purposely vulnerable setup in a safe environment. Real systems often block such inputs using proper sanitisation. Some injections didn’t work due to extra filters still being active. Destructive commands like drop table were noted but not executed. Still, the lab clearly showed how SQL injection works and why input handling matters.