Hacettepe University
Computer Engineering

BBM 459
Secure Programming Laboratory
Programming Assignment 3
SQL Injection

Spring 2021

Ahmet Hakan YILDIZ

Cihan KÜÇÜK

1. Installation:

First we have downloaded ***BWAPP*** application from ***sourceforge.net*** and installed. In addition we have downloaded and installed ***XAMPP*** (***Apache***, ***PHP*** and ***MySQL***) on our machine. But an error occurred because of ***mysql*** functions. Our ***XAMPP*** has a ***PHP*** of version 8 which does not support ***mysql*** functions. It supports ***mysqli*** instead. But ***bWAPP*** application is written years ago and it uses ***mysql*** function calls. There were 2 options. Whether we had to change all ***mysql*** calls to ***mysqli*** calls or install ***bee-box*** virtual machine. We chose installing the virtual machine.

2. Experiment Steps:

After installing ***Oracle VirtualBox*** and installing ***bee-box*** into it, we have reached vulnerable application page using ***Firefox***.

1. SQL Injection (GET/Select)

a) Finding column number

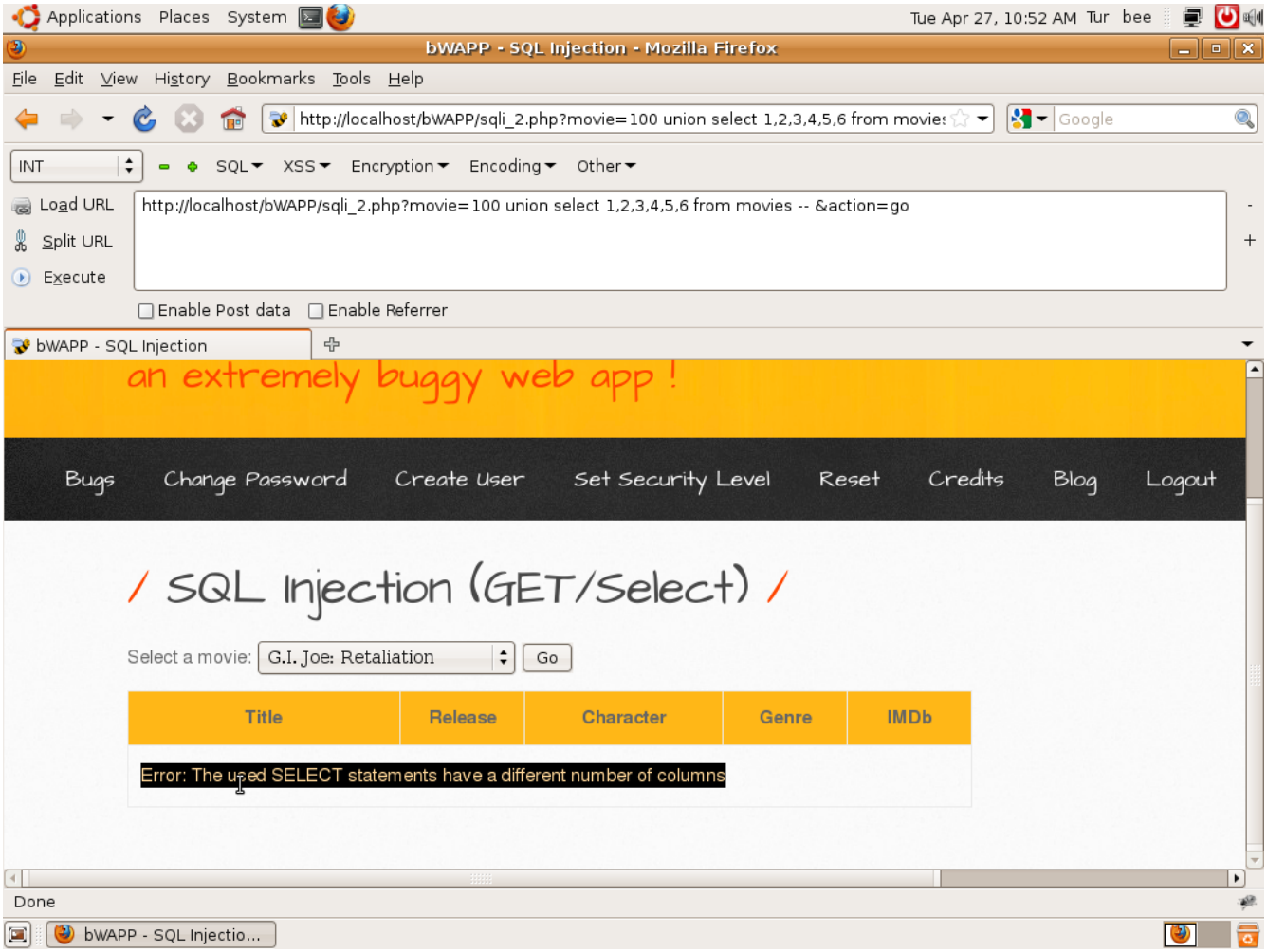
Selecting different movies change id on the URL. We send ***100*** as movie id because 100th movie does not exist in database. After 100, we added ***UNION*** and other ***SQL*** commands that we want. We add -- at the end as a SQL comment.

Number of columns is found by using ***UNION*** operation. For ***UNION*** operation, two tables must have same number of columns. We try union with arbitrary number of columns.

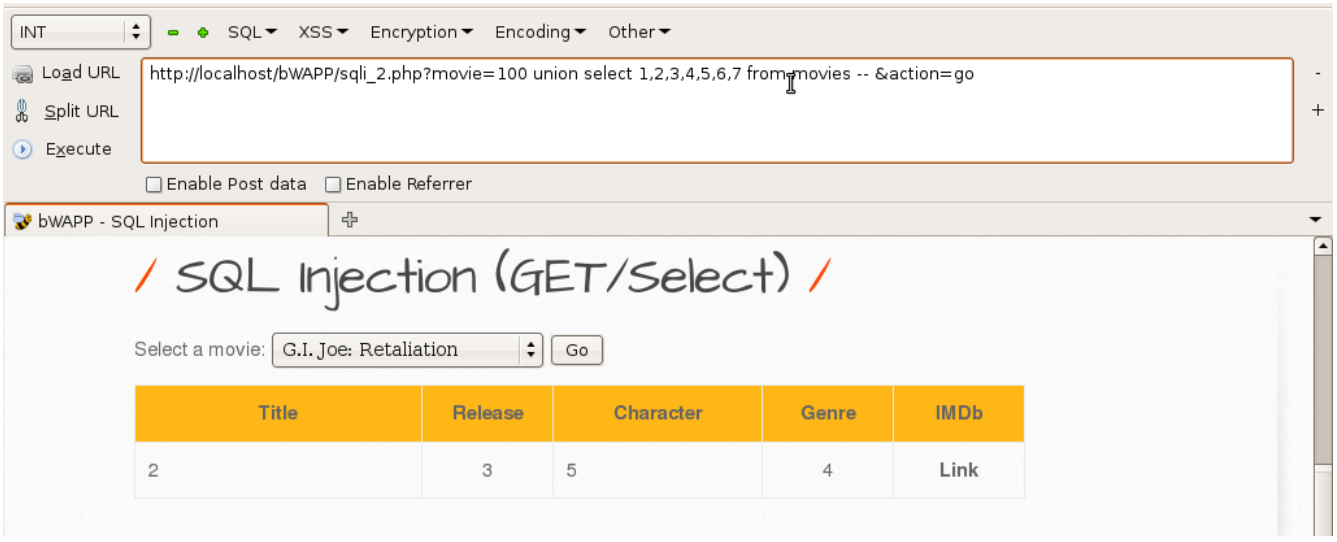
If number of columns are not matched, following error is created.

Error: The used SELECT statements have a different number of columns.

Hence we continued trying different number of columns until there is no error.



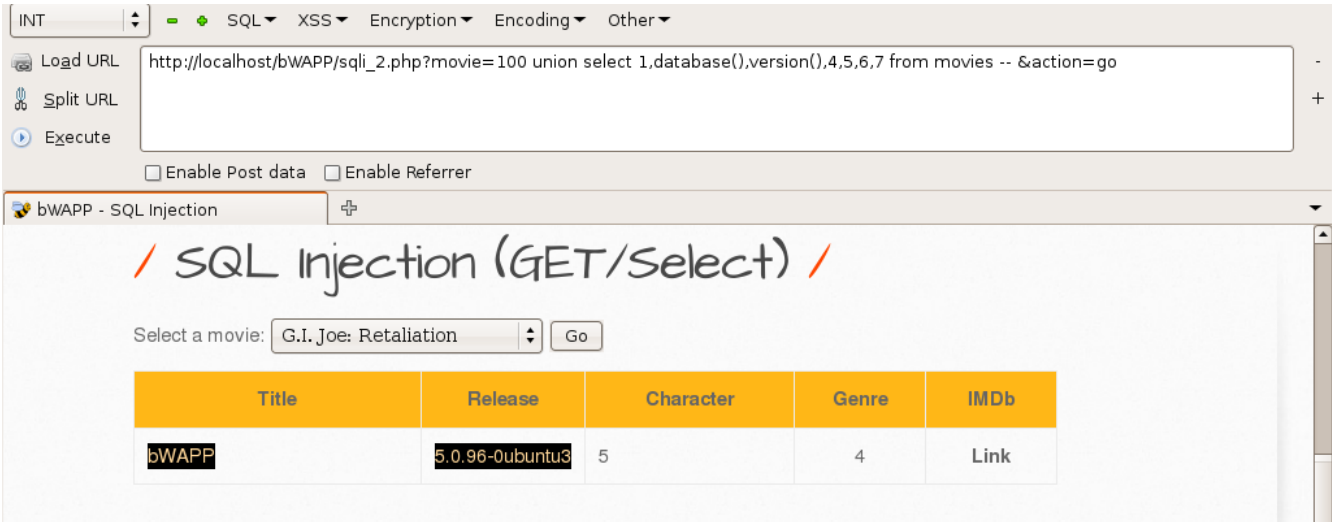
6 columns gave error, so we continued increasing number of columns.



After testing, it is found that there are 7 columns.

b-c) Finding database name and database version

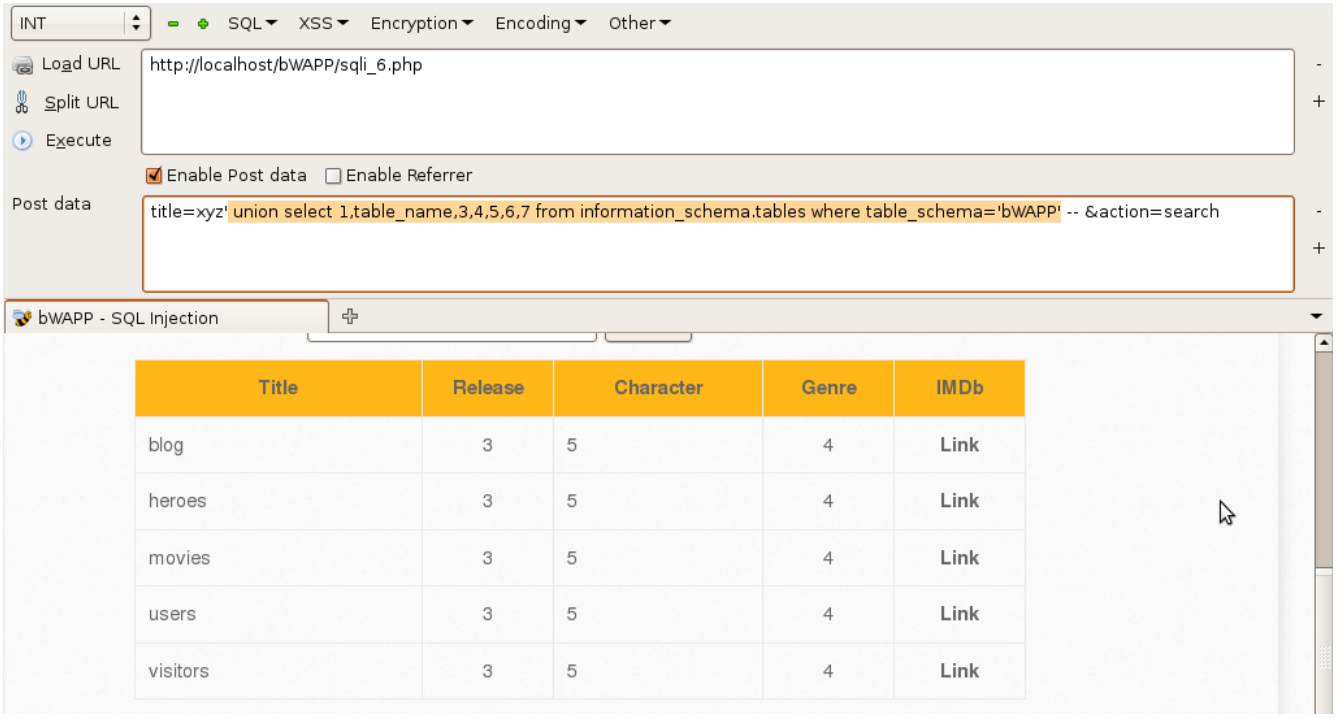
We insert ***database()*** instead of 2 and ***version()*** instead of 3 in ***select*** statement. That request returns database name as ***bWAPP***. and version as ***5.0.96-0ubuntu3***.



2. SQL Injection (POST/Search)

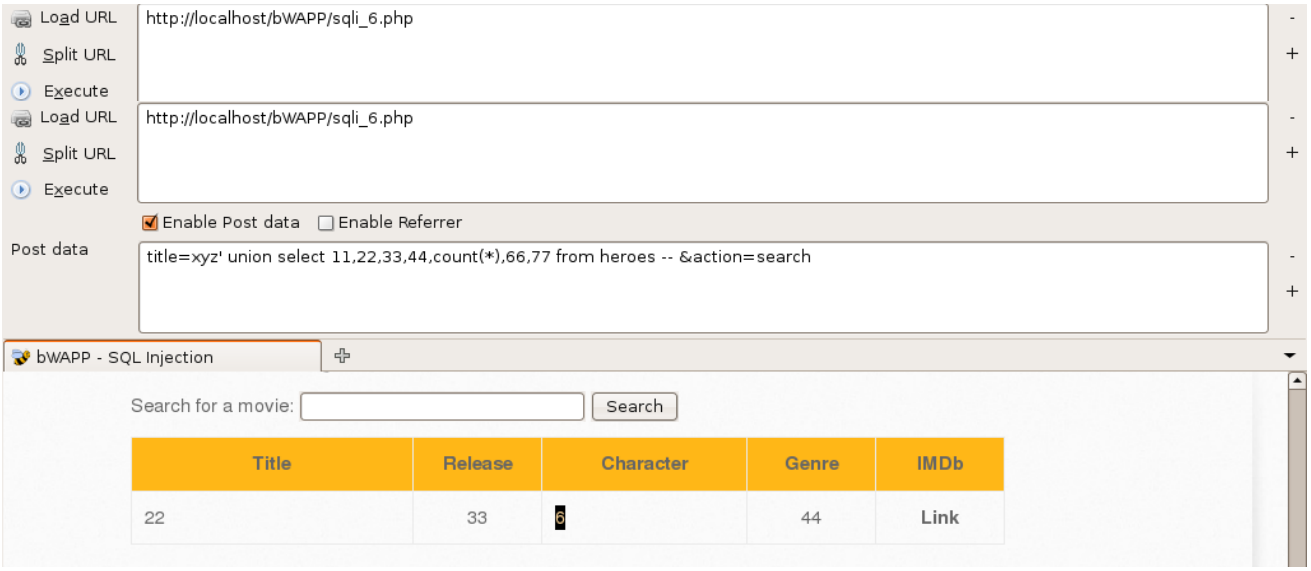
a) Listing table names and number of records

We use post messages instead of get messages. When we query ***information_schema.tables***, we can access table names. But in order to get tables for this specific application, we add ***where table_schema='bWAPP'*** clause in the query. This query returns ***blog***, ***heroes***, ***movies***, ***users*** and ***visitors*** tables.



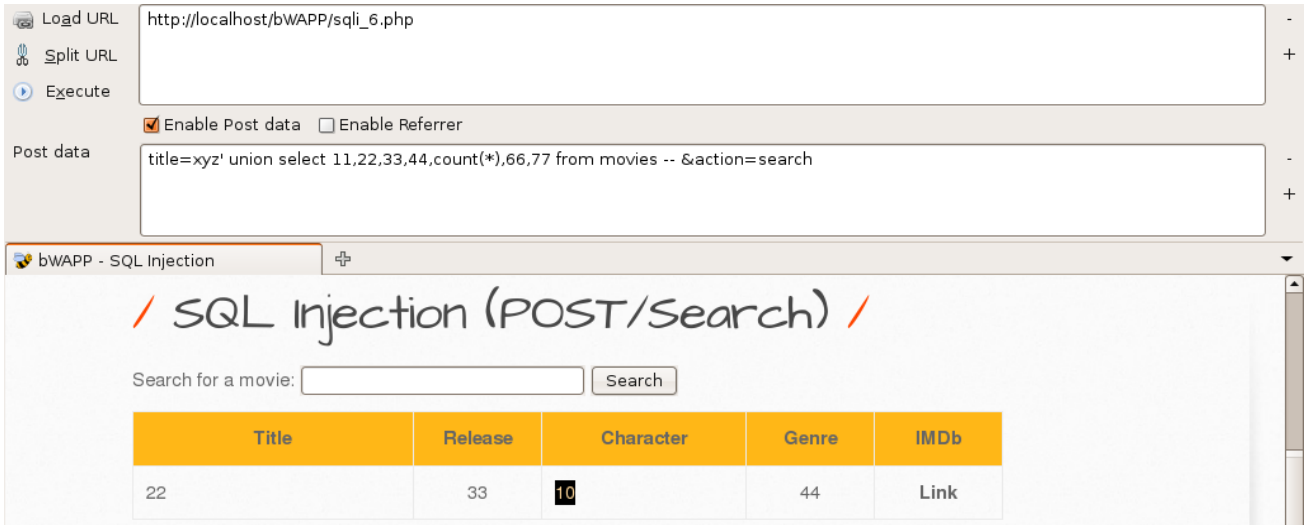
Number of records in each table is retrieved by ***count(*)*** function.

The ***users*** table has 2 records as shown below.

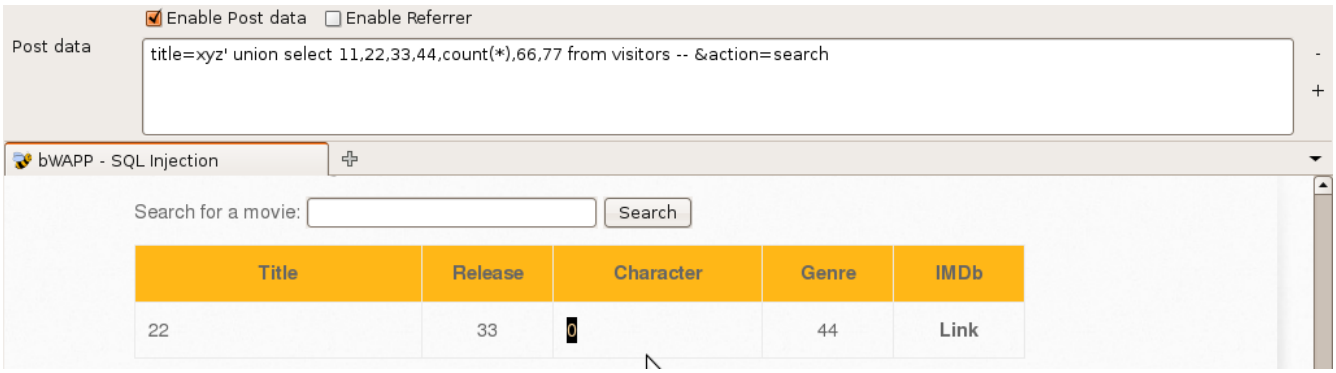


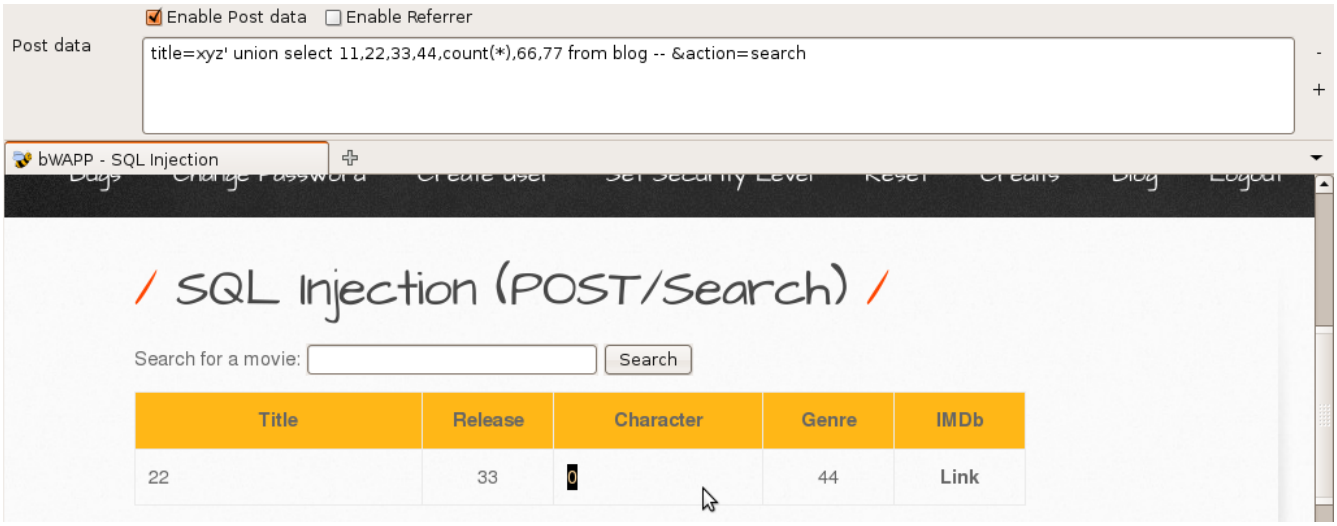
The ***heroes*** table has 6 records as shown below.

The ***movies*** table has 10 records as shown below.



The ***visitors*** and ***blog*** table has no records as shown below

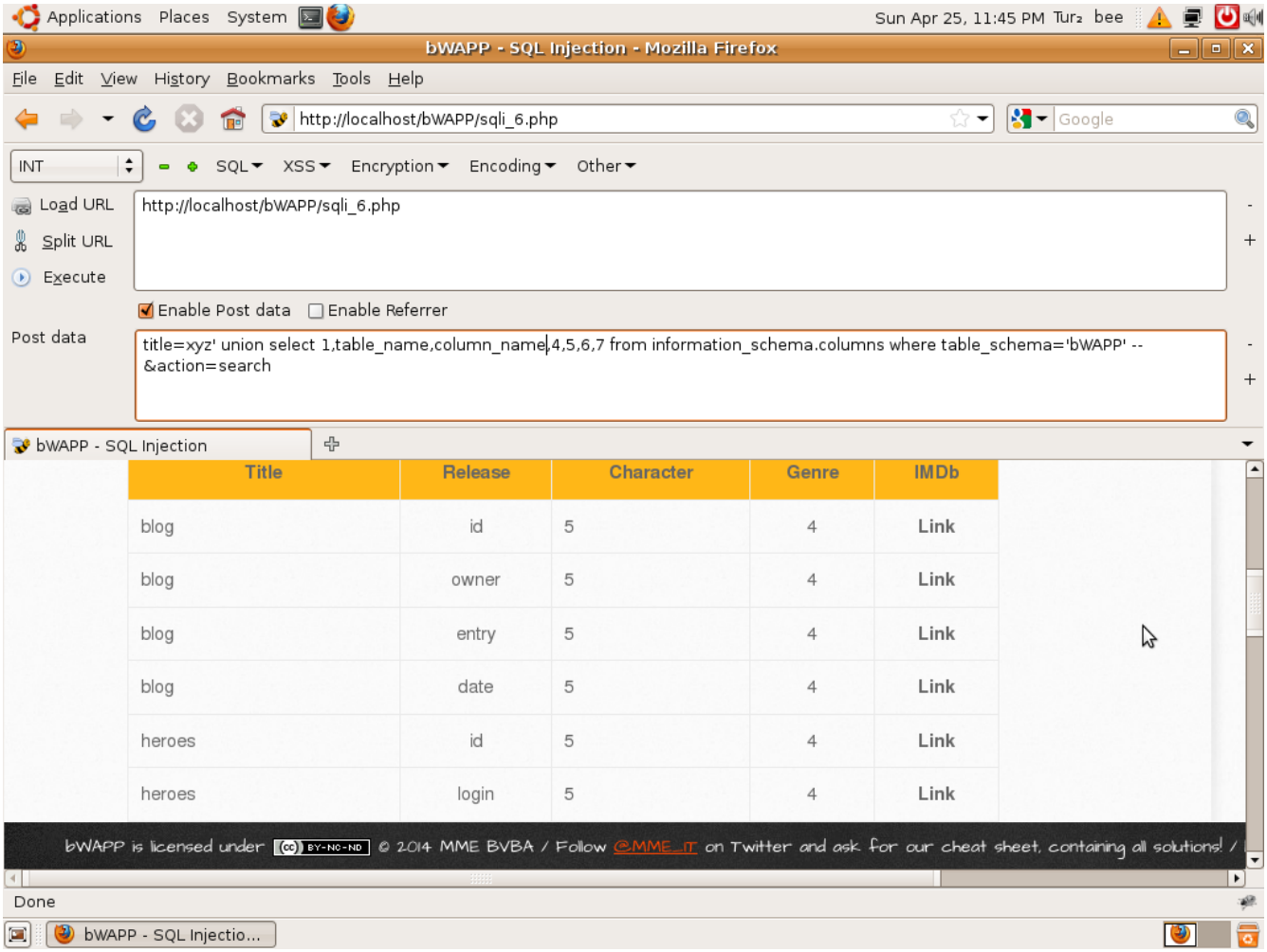




b) Listing column names

When using ***information_schema.columns***, ***table_name*** and ***column_name*** gives table names and column names. Here are the columns of the corresponding tables:

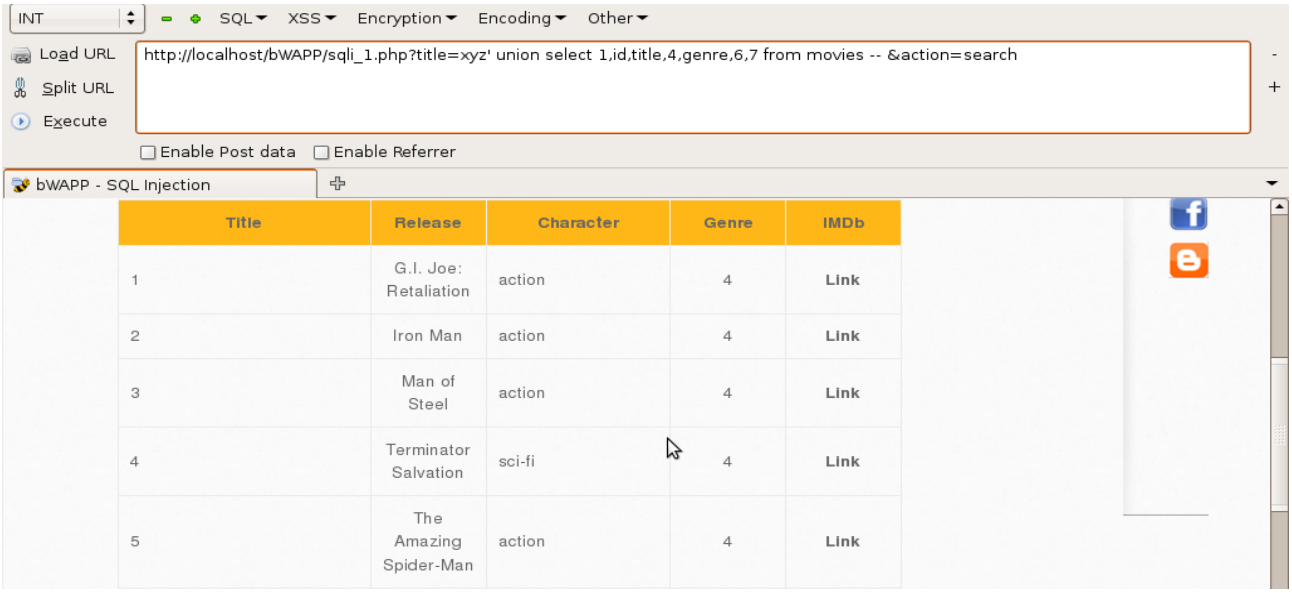
Table Name	**Column Names**
blog	id, owner, entry, date
heroes	id, login, password, select
movies	id, title, release_date, genre, main_character, imdb, tickets_stock
users	id, login, password, email, secret, activation_code, activated, reset_code, admin, uid, name, pass, mail, theme, signature_format, created, access, status, timezone, language, picture
visitors	id, ip_address, user_agent, date



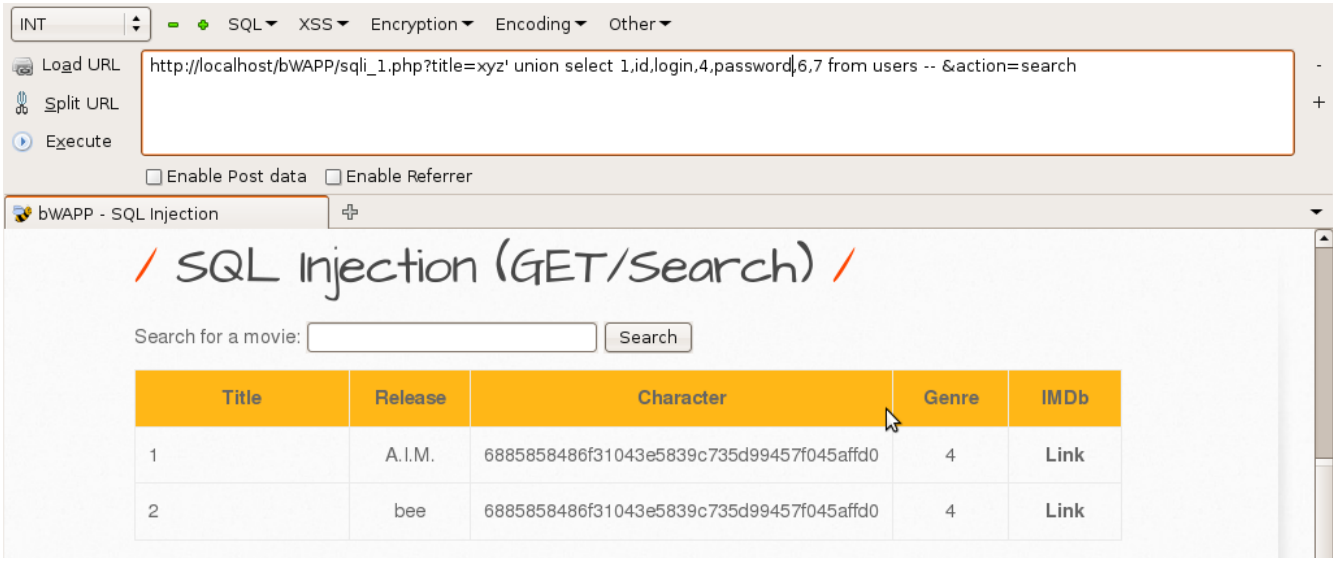
3. SQL Injection (GET/Search)

a) Listing all records in each table

We used ***select 1,id,title,4,genre,5,6,7 from movies*** query in order to list all records in the movies table.



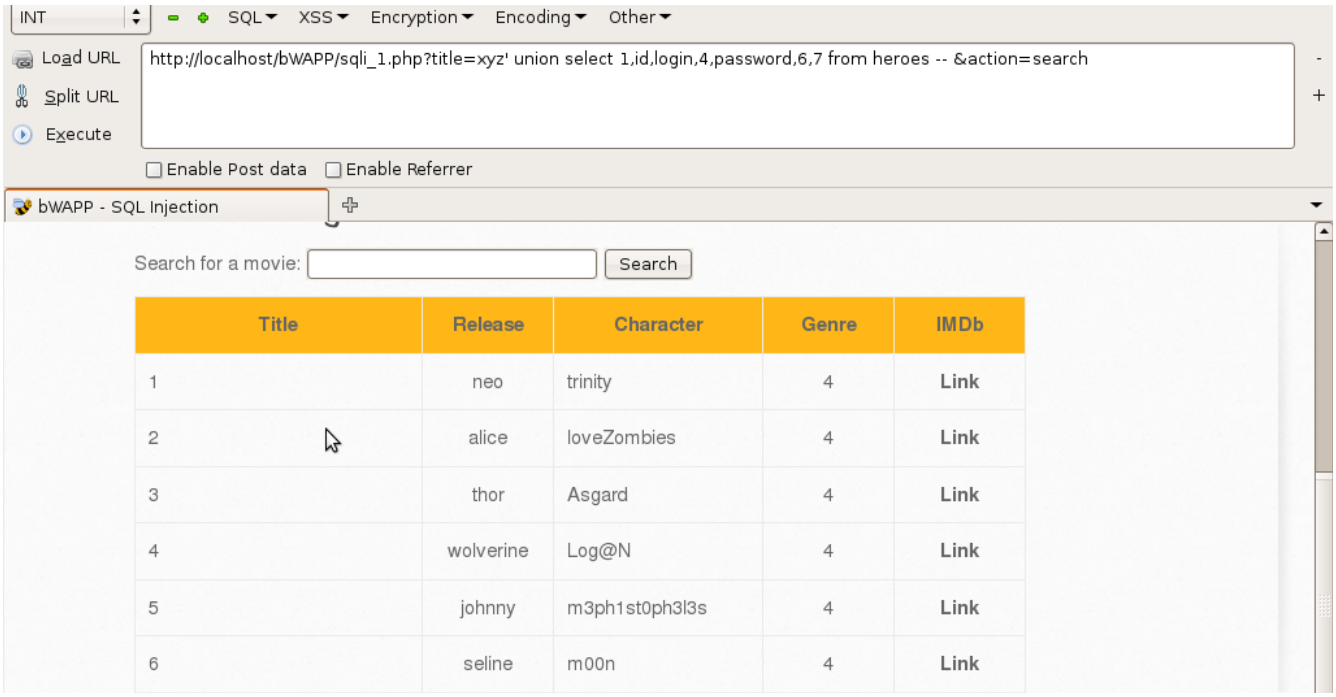
We used ***select 1,id,login,4,password,6,7 from users*** query in order to list all records in the users table.



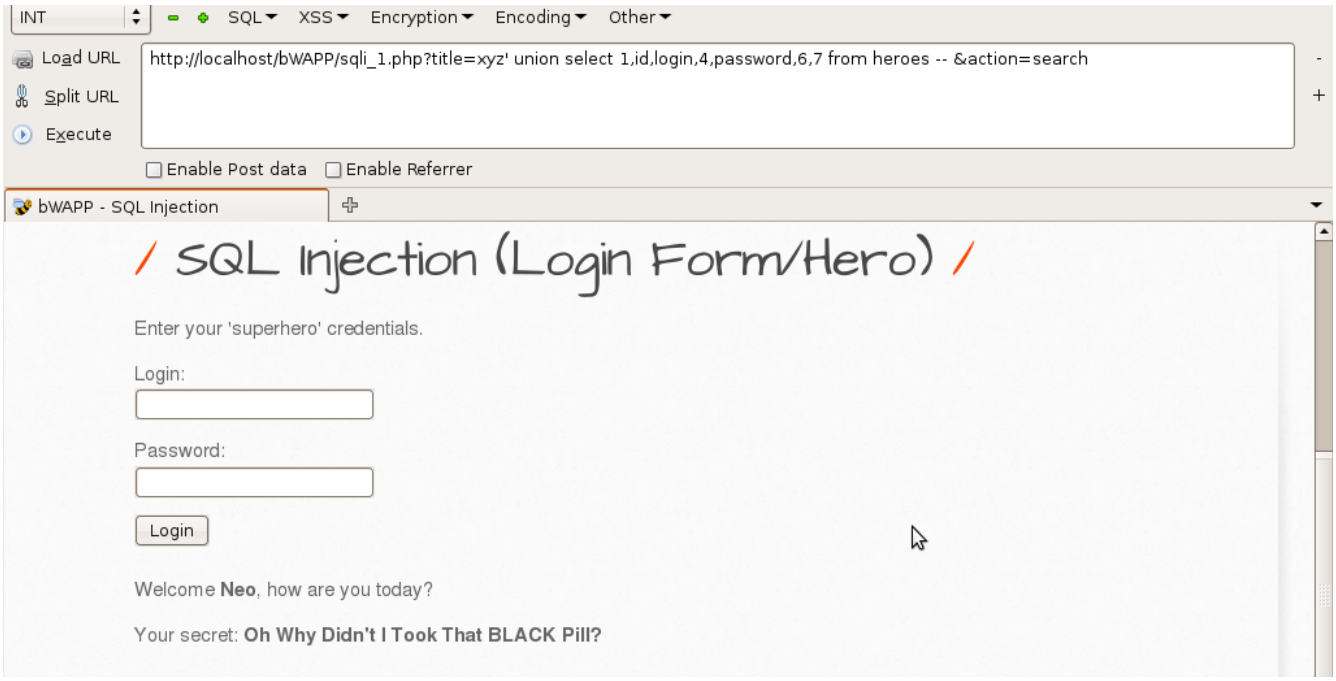
The content of ***blog*** and ***visitors*** table turned out to be empty.

b) Getting credentials of a superhero

We printed credentials(***id, login, password*** fields) of superheroes.

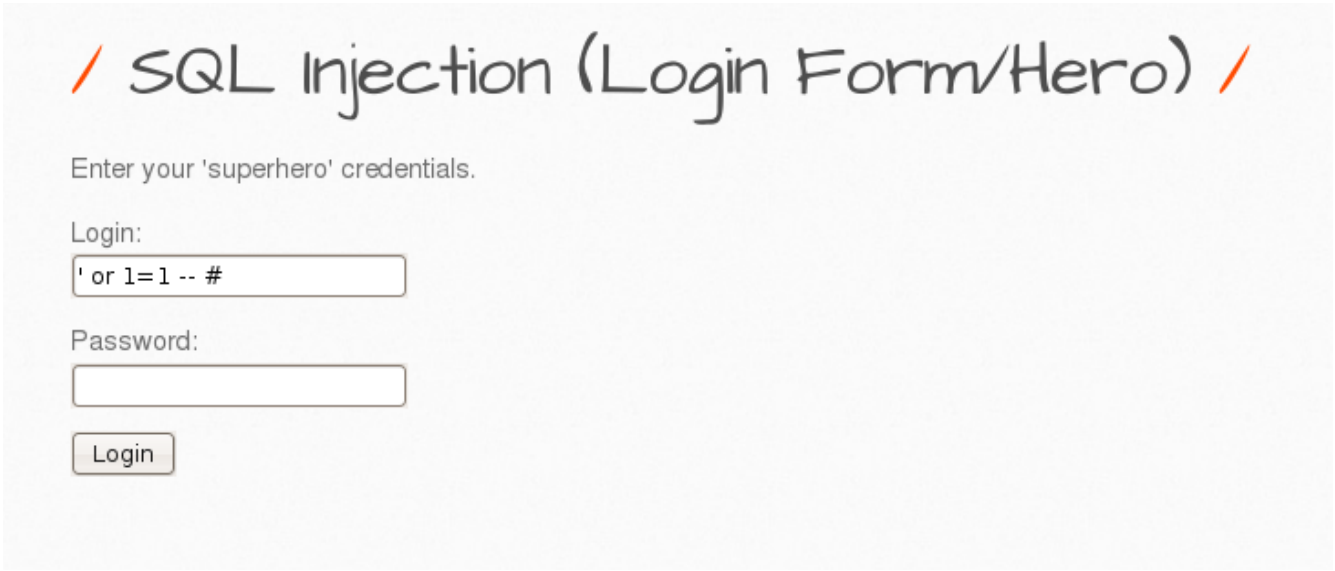


Then we learn ***neo***'s password is ***trinity*** and used it to login as ***neo***.

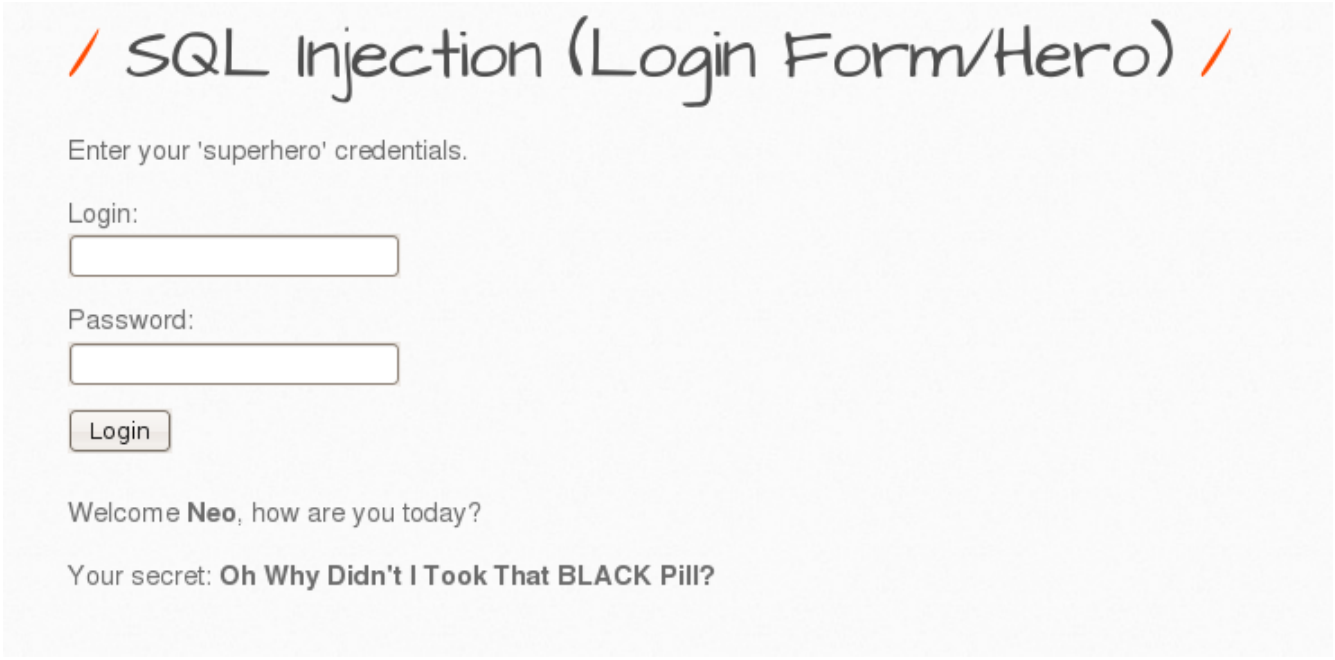


c) Repeating the previous step

We know there are SQL vulnerabilities on this site. That's why we carry out our direct attack. When we type ' or 1=1 -- # in the Username field, we will be manipulating the SQL vulnerability and logging in with the Neo user.



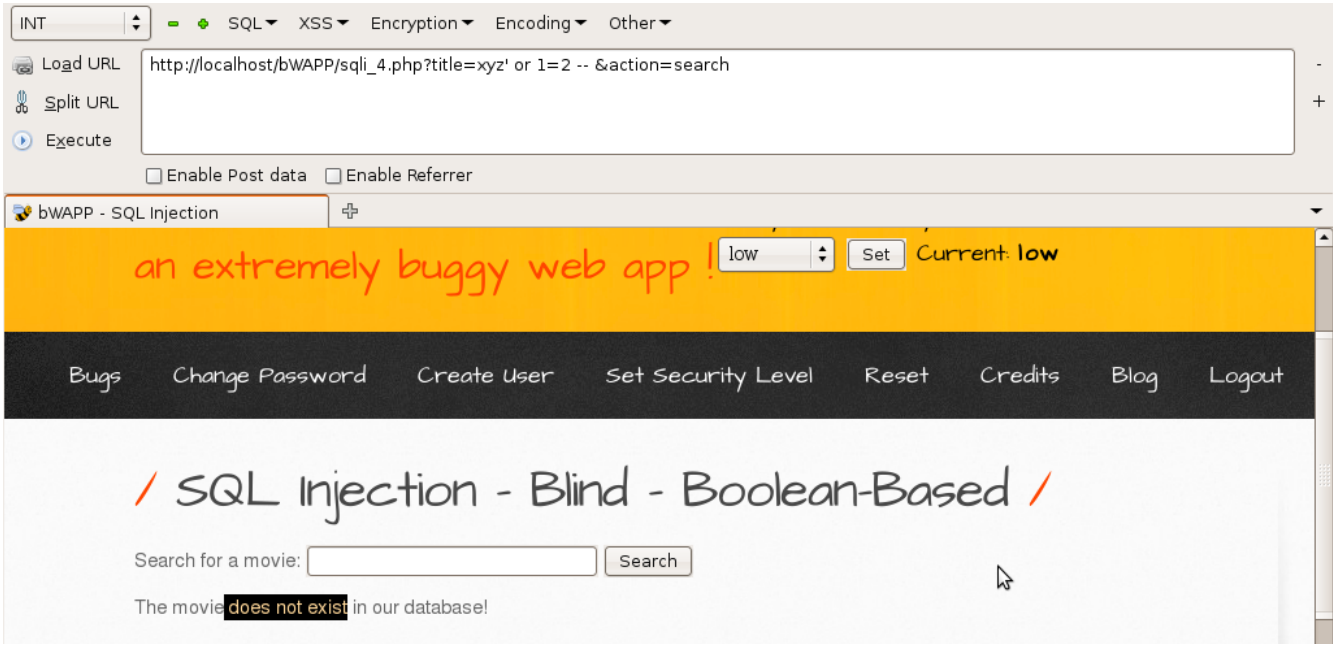
We type ‘ or 1=1 -- # to username field. Password field is not important. After that:



We found a reference[1] for a more detailed explanation on this topic. You can find it in the References section.

4. SQL Injection - Blind - Boolean-Based

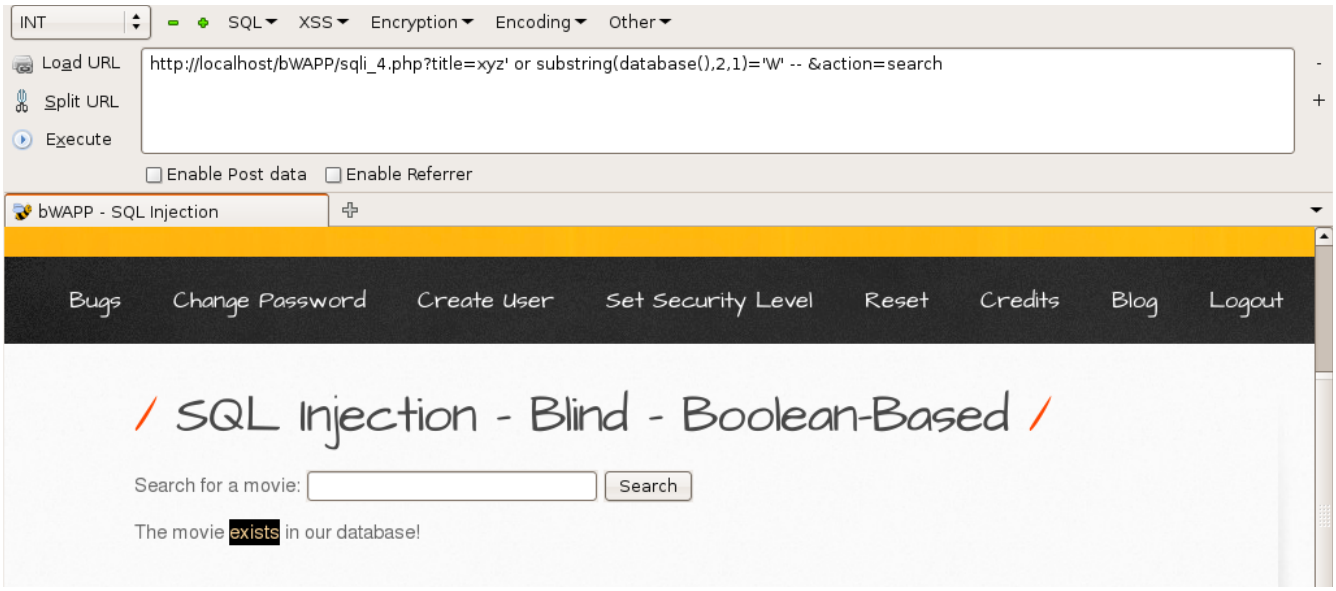
We add or condition to our query. If the result of the condition is true, page returns movie exists else page returns movie does not exist message. Using this message, we can verify length of variables by ***length()*** method and characters of variables by ***substring()*** method.



a) Verifying name of the database

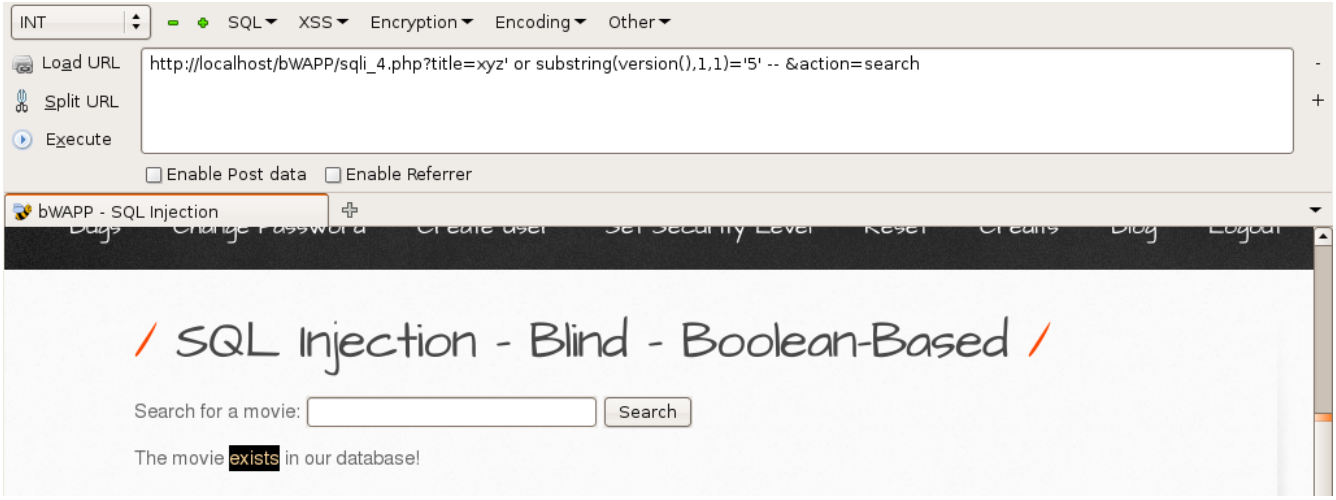
First we verify that length of the database is 5.

Then we verify the characters one by one by using ***substring()*** method.



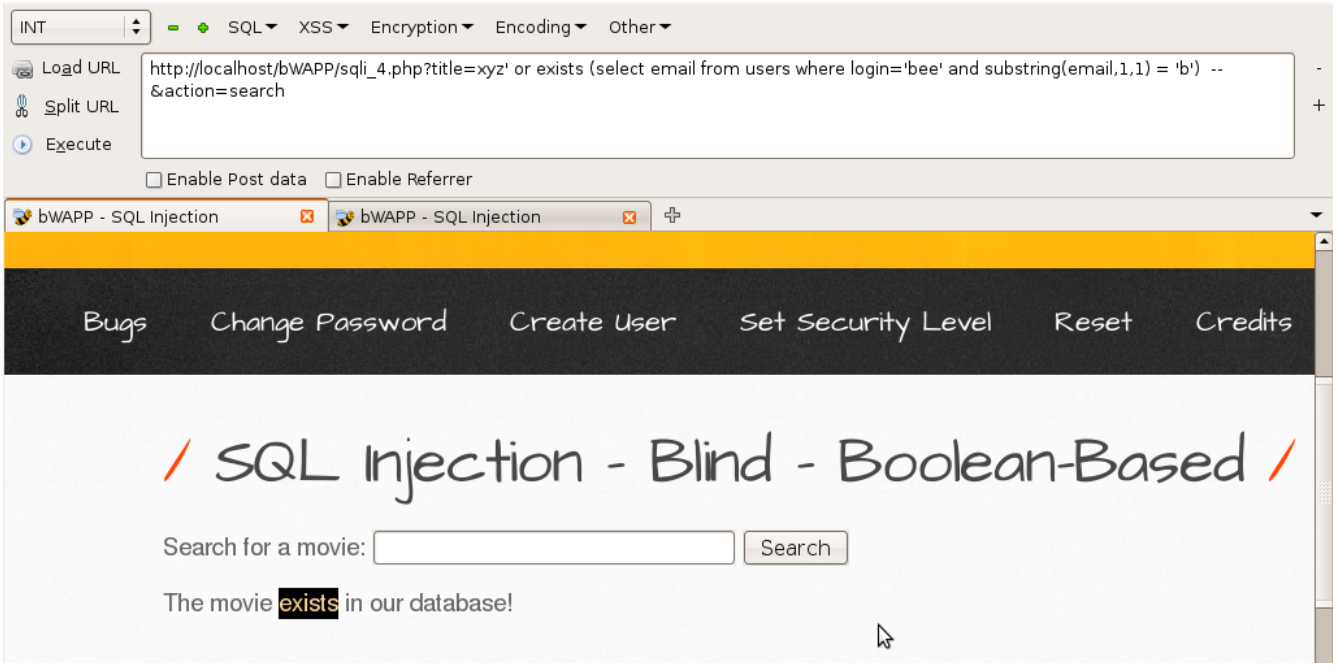
b) Verifying version of the database

Similar to the previous part, we used again the ***length*** and ***substring*** methods in order to verify version of the database.



c) Verifying e-mail address

We used ***exists*** function with select clause. With the help of ***substring*** method all characters in the email can be verified easily.



References

[1] Login Bypass Using SQL Injection, http://www.securityidiots.com/Web-Pentest/SQL-Injection/bypass-login-using-sql-injection.html