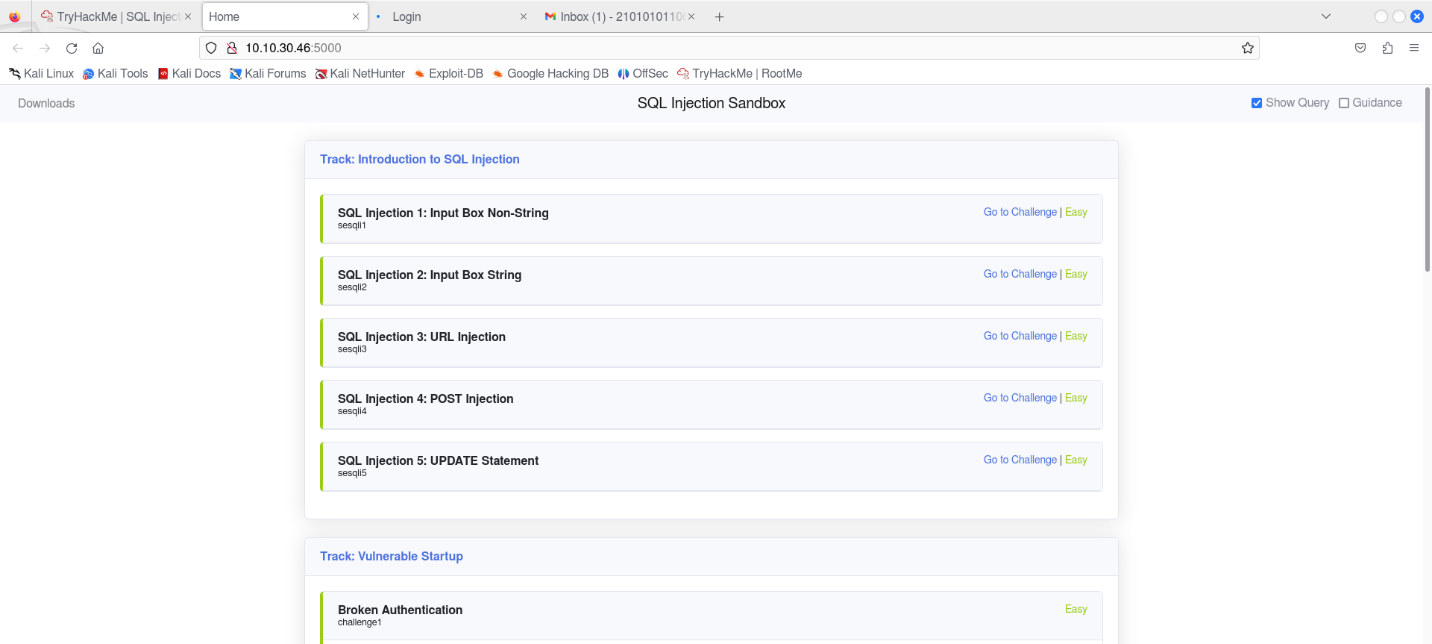
**Lab Practical #06:**

TryHackMe Room: - **SQL Injection Lab**

1. **Introduction**

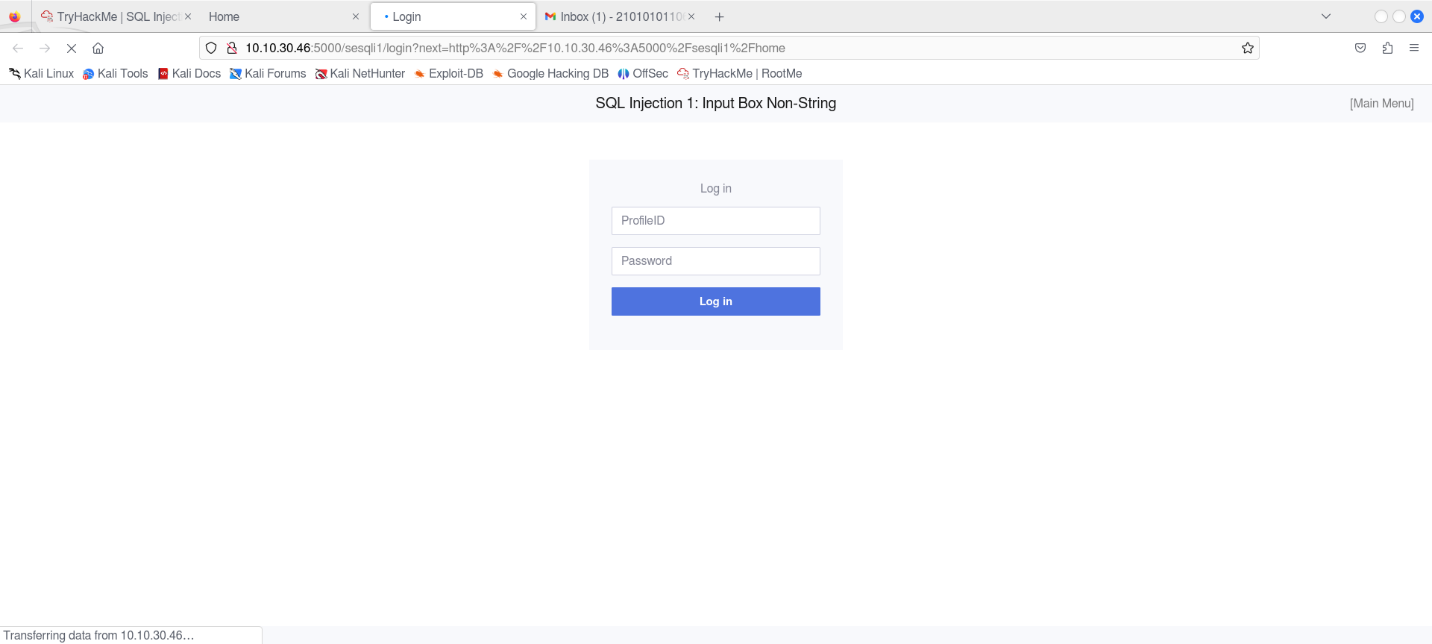
This room is meant as an introduction to SQL injection and demonstrates various SQL injection attacks.

The web application can be found at: [http://10.10.30.46:5000](http://10.10.30.46:5000/" \t "_blank)



1. **Introduction to SQL Injection: Part 1**

**SQL Injection 1: Input Box Non-String**

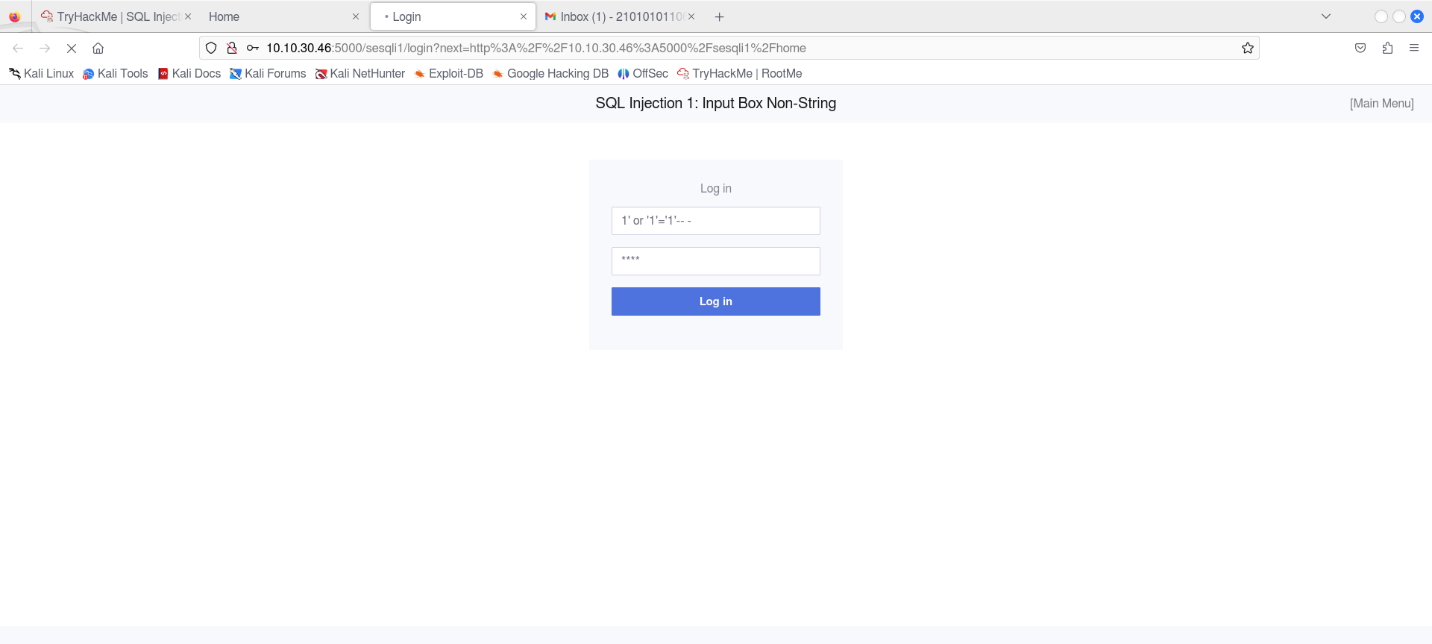


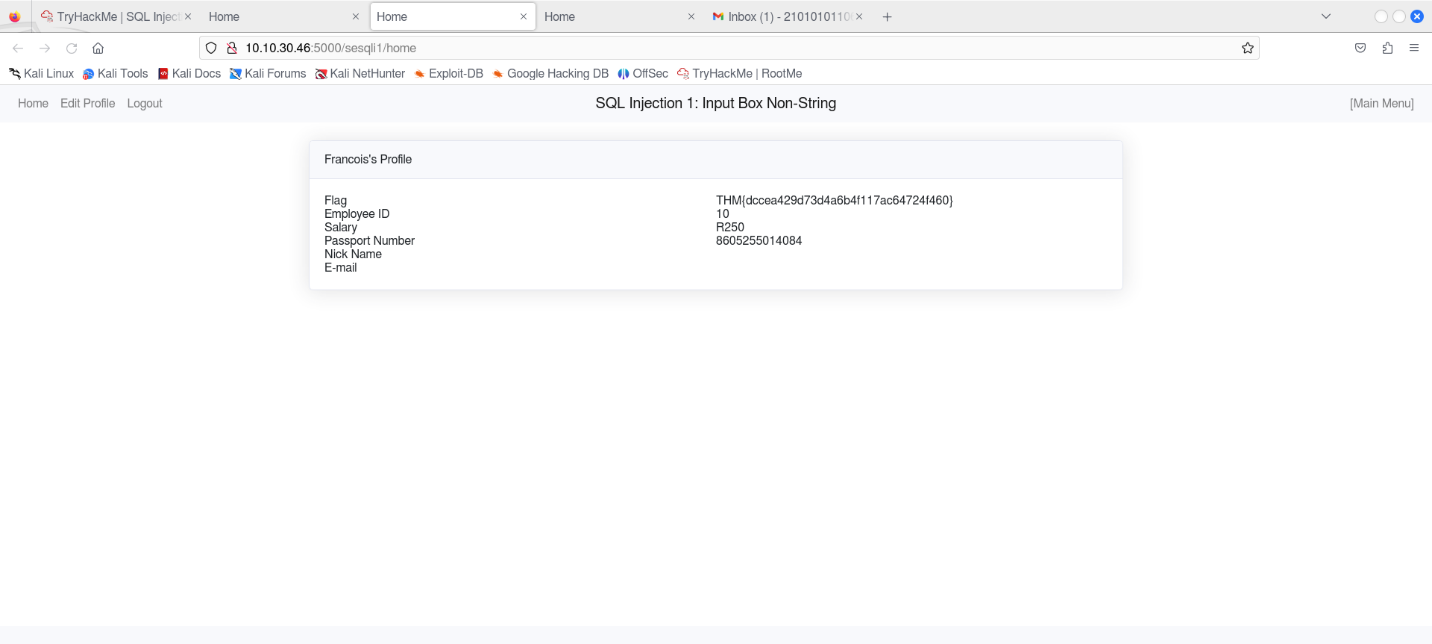
When a user logs in, the application performs the following query:

*SELECT uid, name, profileID, salary, passportNr, email, nickName, password FROM usertable WHERE profileID=10 AND password = 'ce5ca67...'*

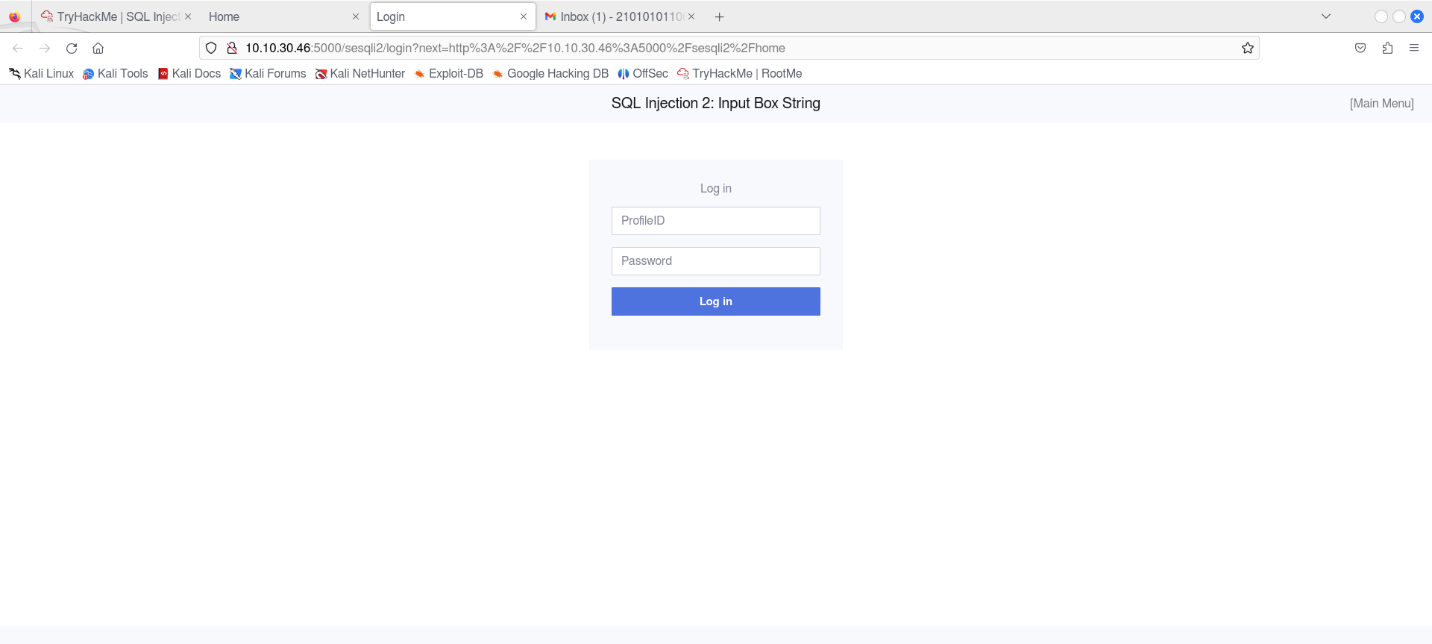
Since there is no input sanitization, it is possible to bypass the login by using any True condition such as the one below as the ProfileID

1 or 1=1-- -





**SQL Injection 2: Input Box String**

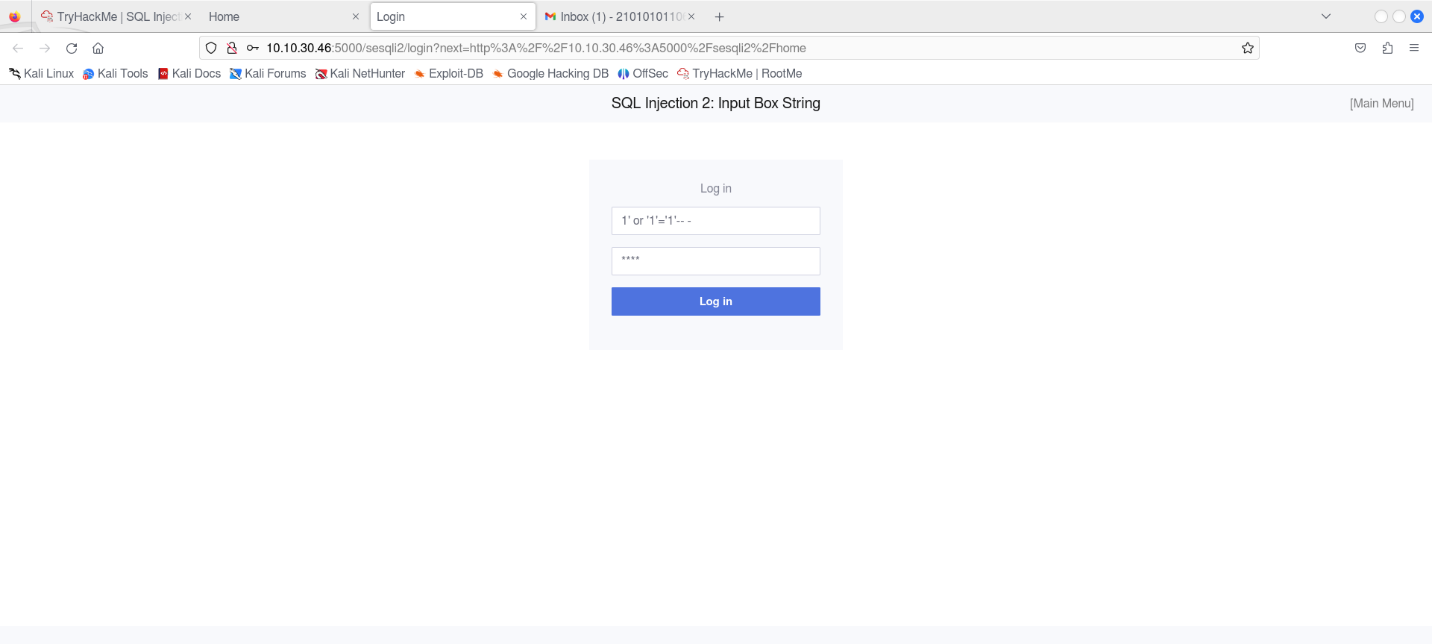


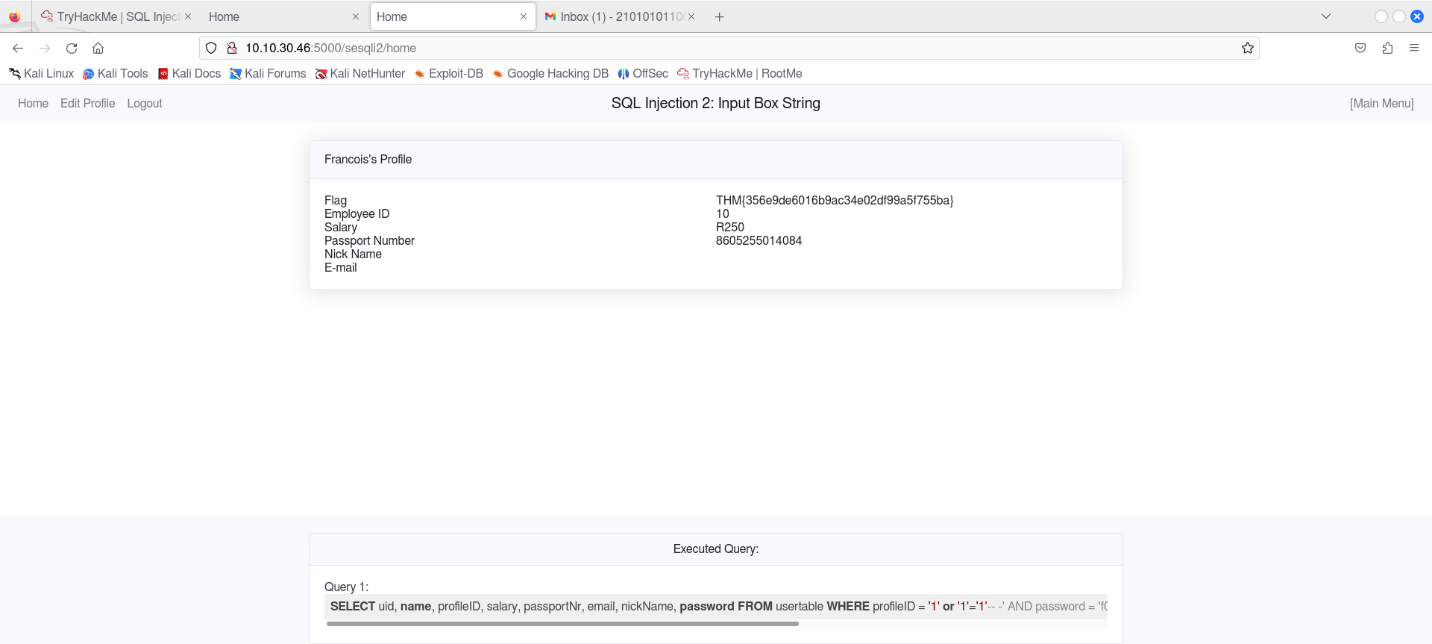
This challenge uses the same query as in the previous challenge. However, the parameter expects a string instead of an integer, as can be seen here:

*ProfileID=’10’*

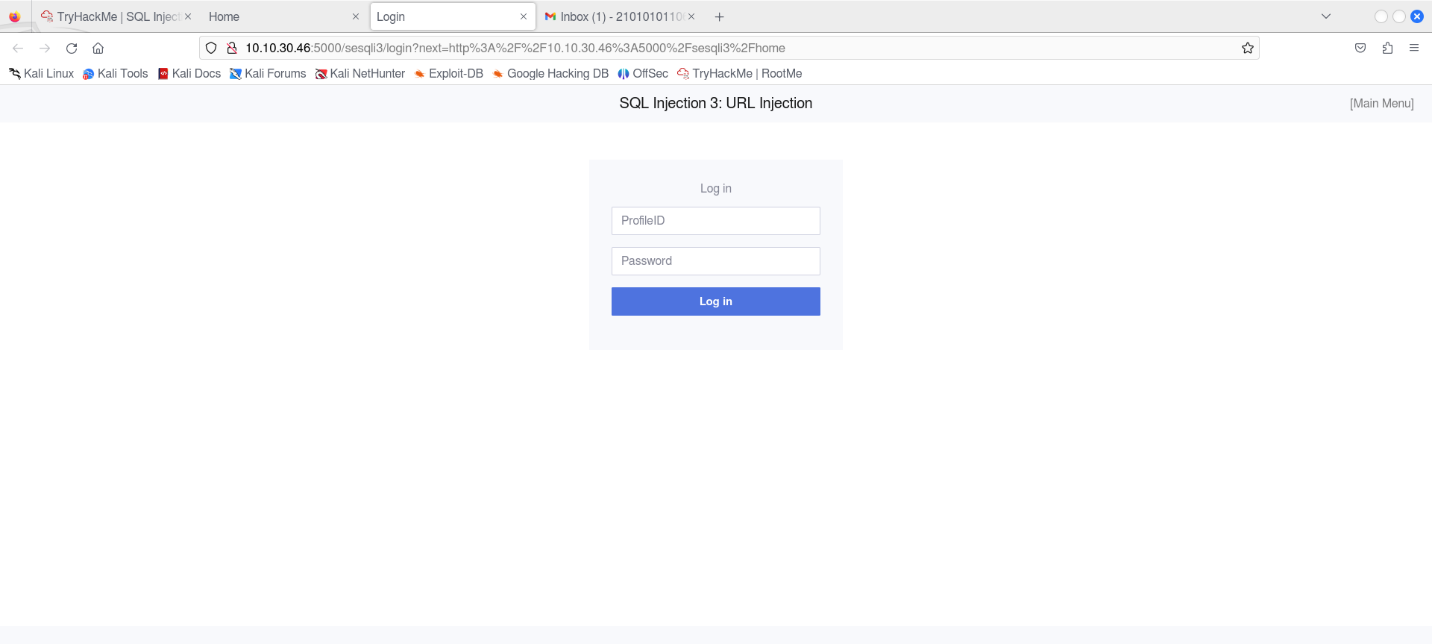
Since it expects a string, we need to modify our payload to bypass the login slightly. The following line will let us in:

1 or 1=1-- -





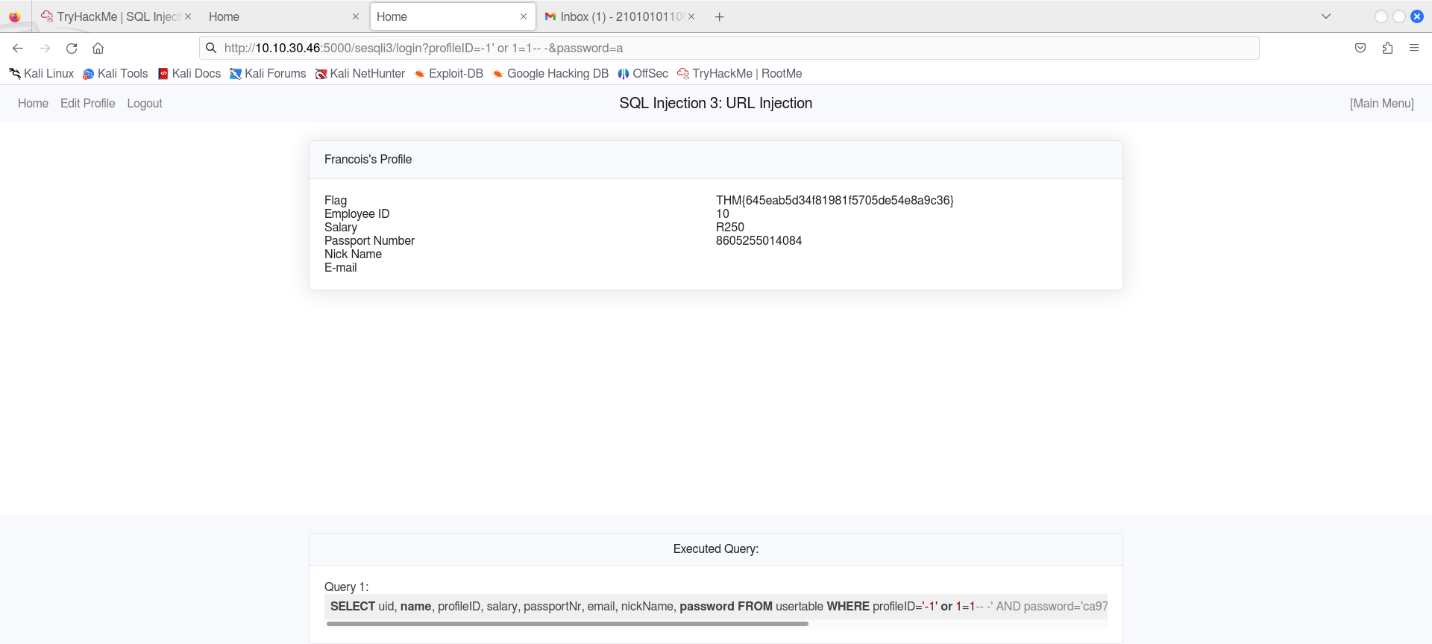
**SQL Injection 3: URL Injection**



Here, the SQL query is the same as the previous one, But in this case, the malicious user input cannot be injected directly into the application via the login form because some client-side controls have been implemented

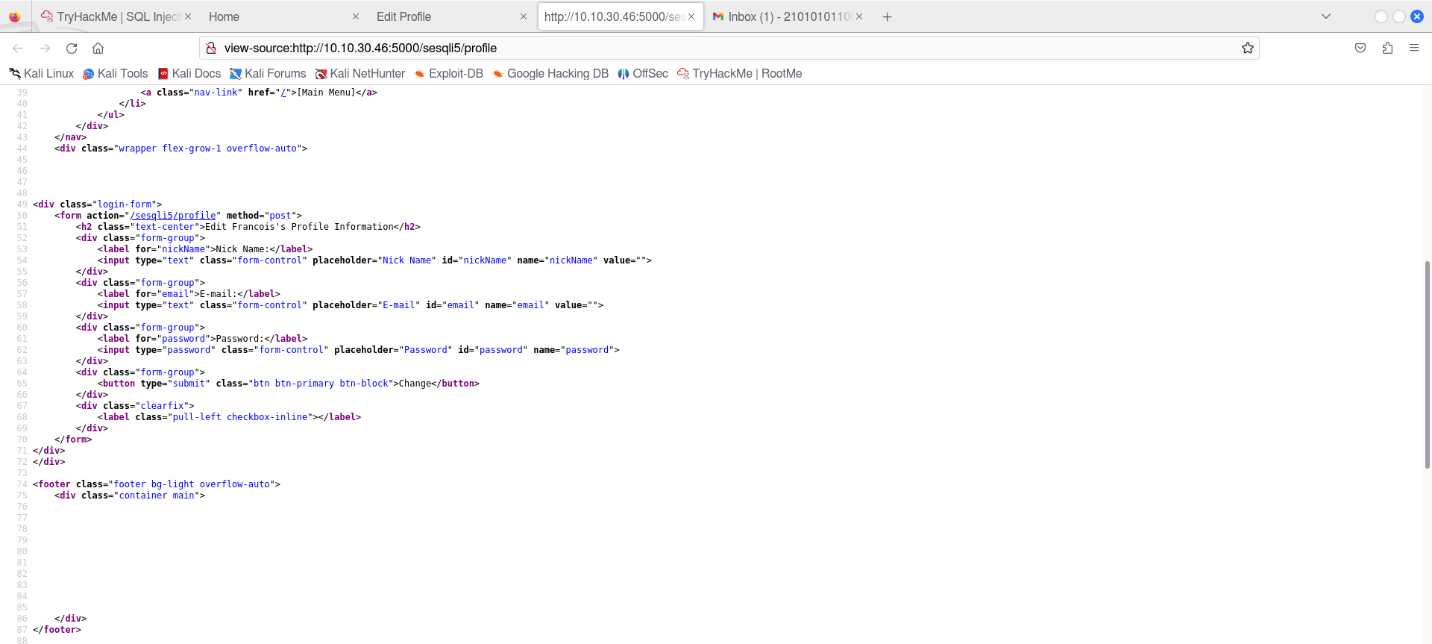
The browser will automatically urlencode this for us. Urlencoding is needed since the HTTP protocol does not support all characters in the request. When urlencoded, the URL looks as follows:

<http://10.10.30.46:5000/sesqli3/login?profileID=-1%27%20or%201=1--%20-&password=a>



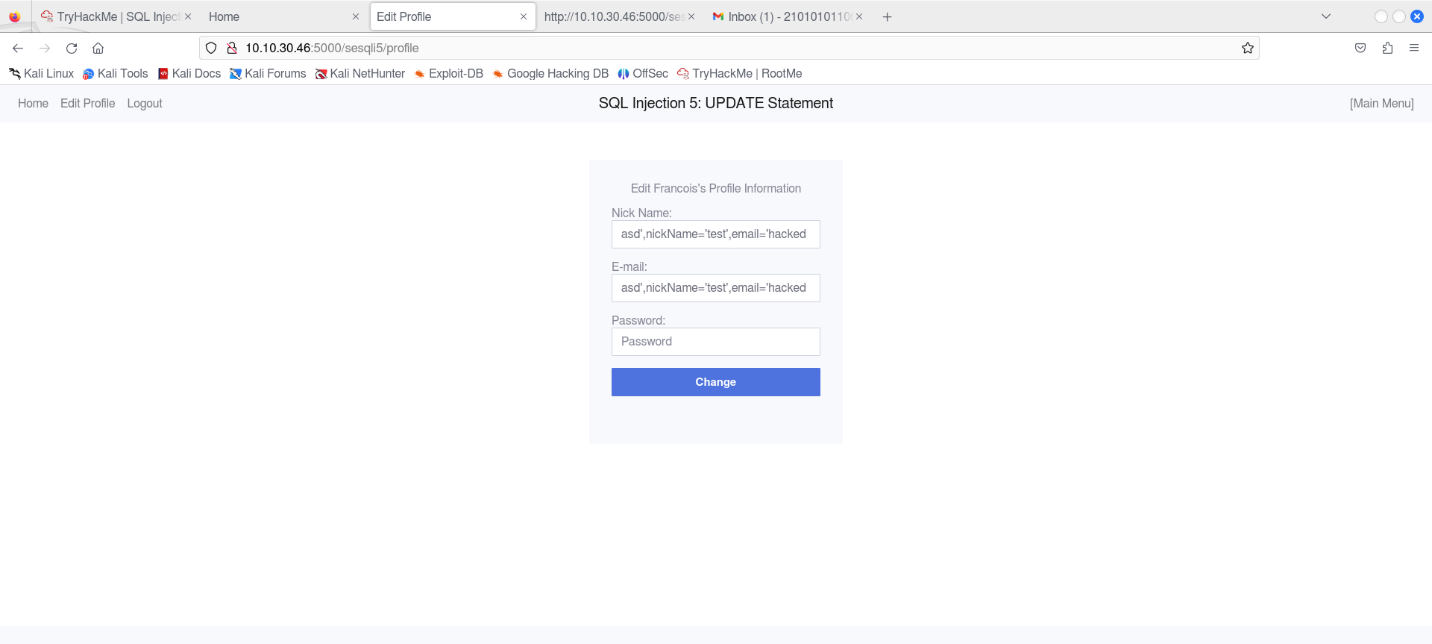
**Introduction to SQL Injection: Part 2**

If a SQL injection occurs on an UPDATE statement, the damage can be much more severe as it allows one to change records within the database. In the employee management application, there is an edit profile page as depicted in the following figure.

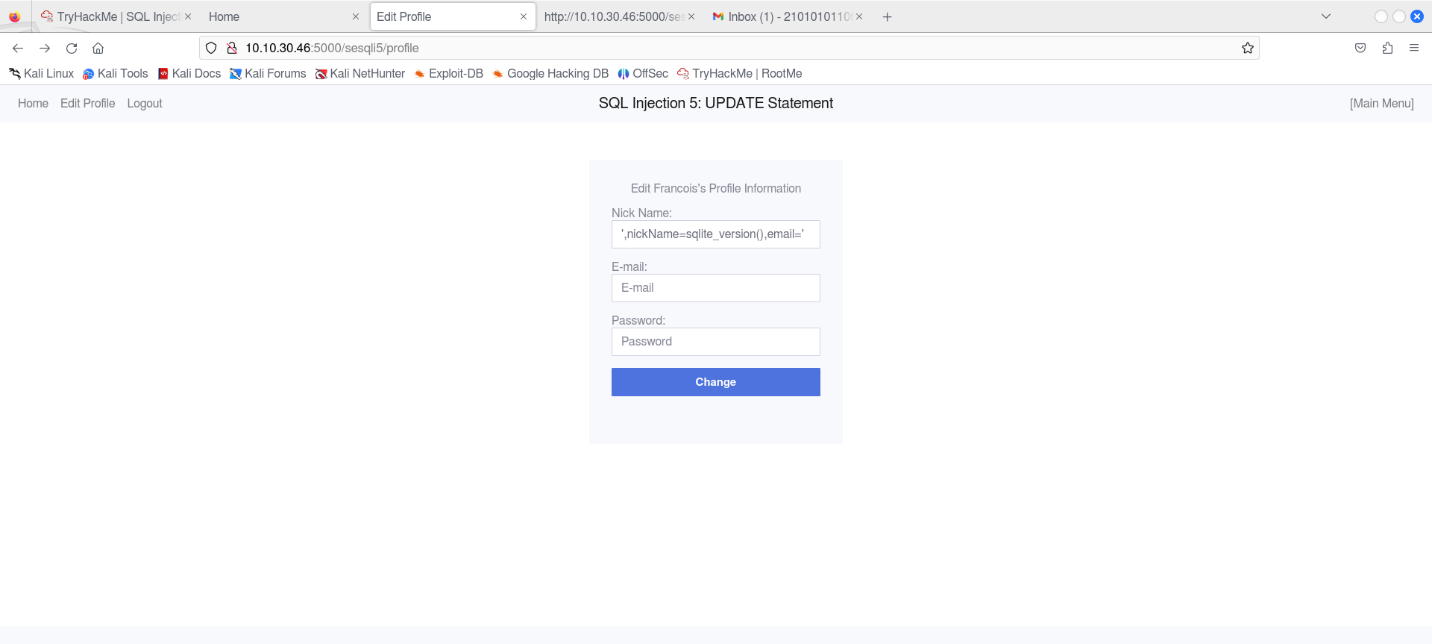


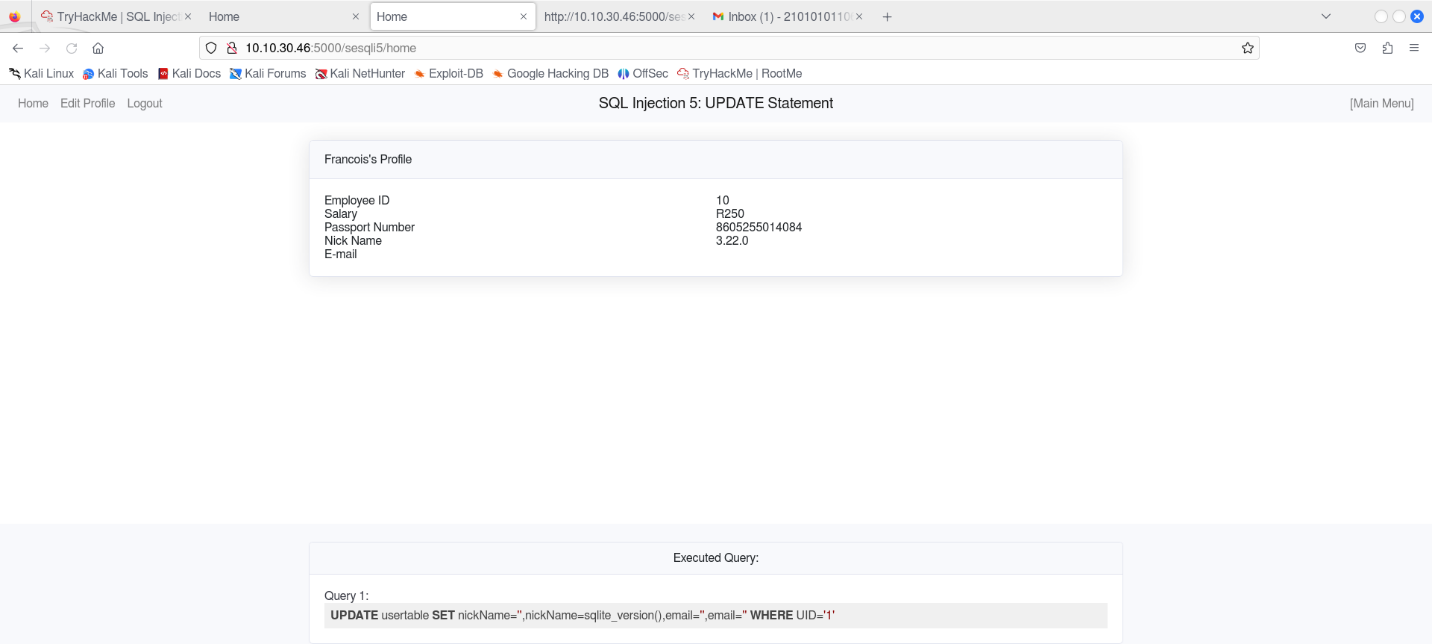
To confirm that the form is vulnerable and that we have working column names, we can try to inject something similar to the code below into the nickName and email field:

asd',nickName='test',email='hacked

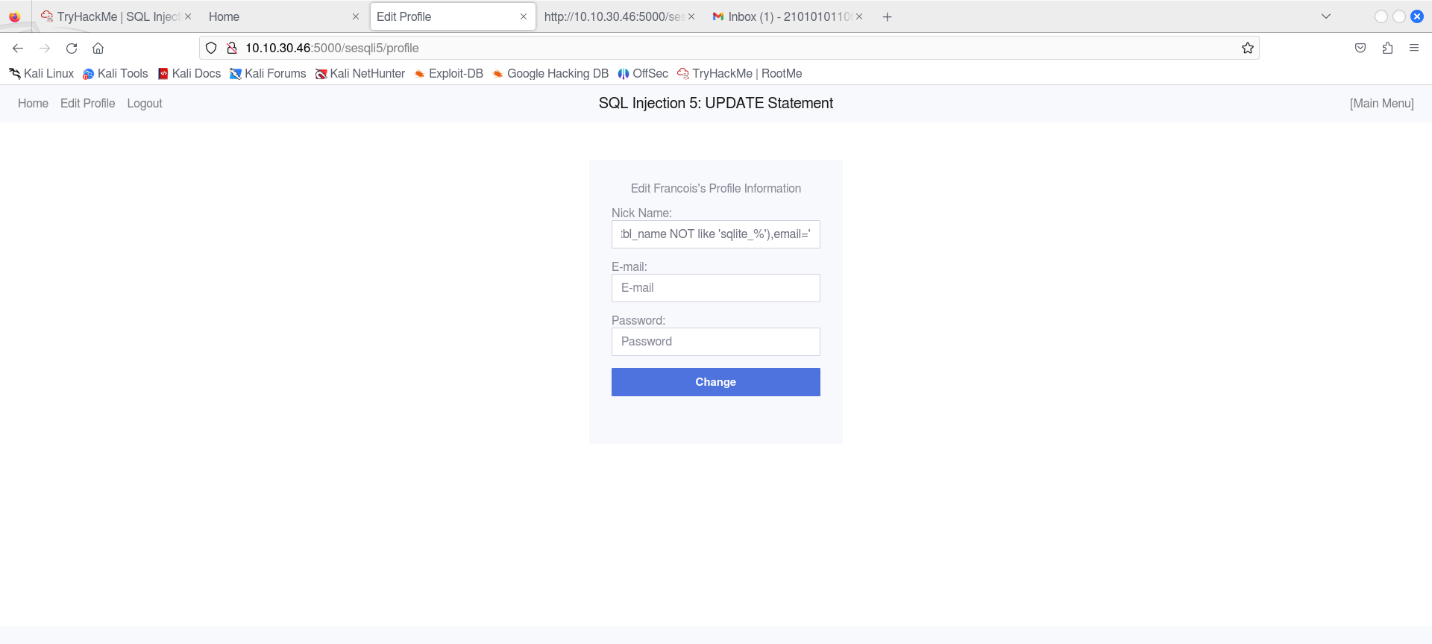


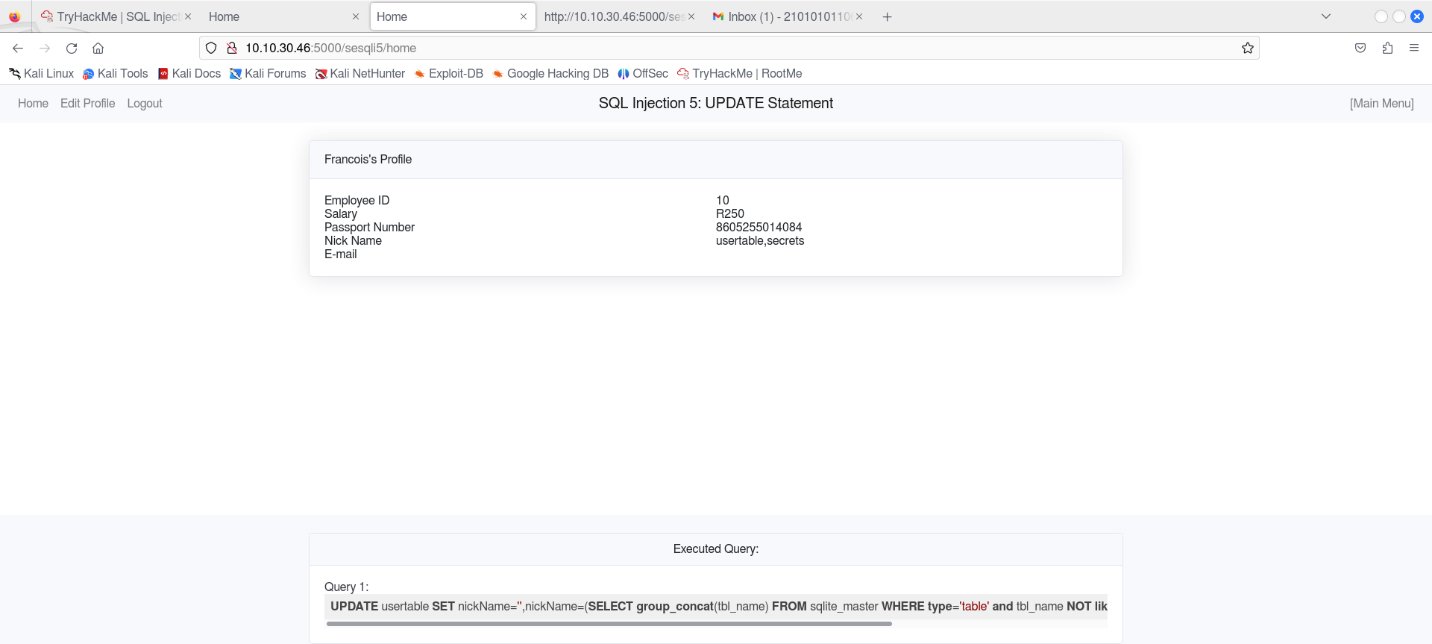
Injecting the line with "sqlite\_version()" into the nickName field shows that we are dealing with SQLite and that the version number is 3.27.2:





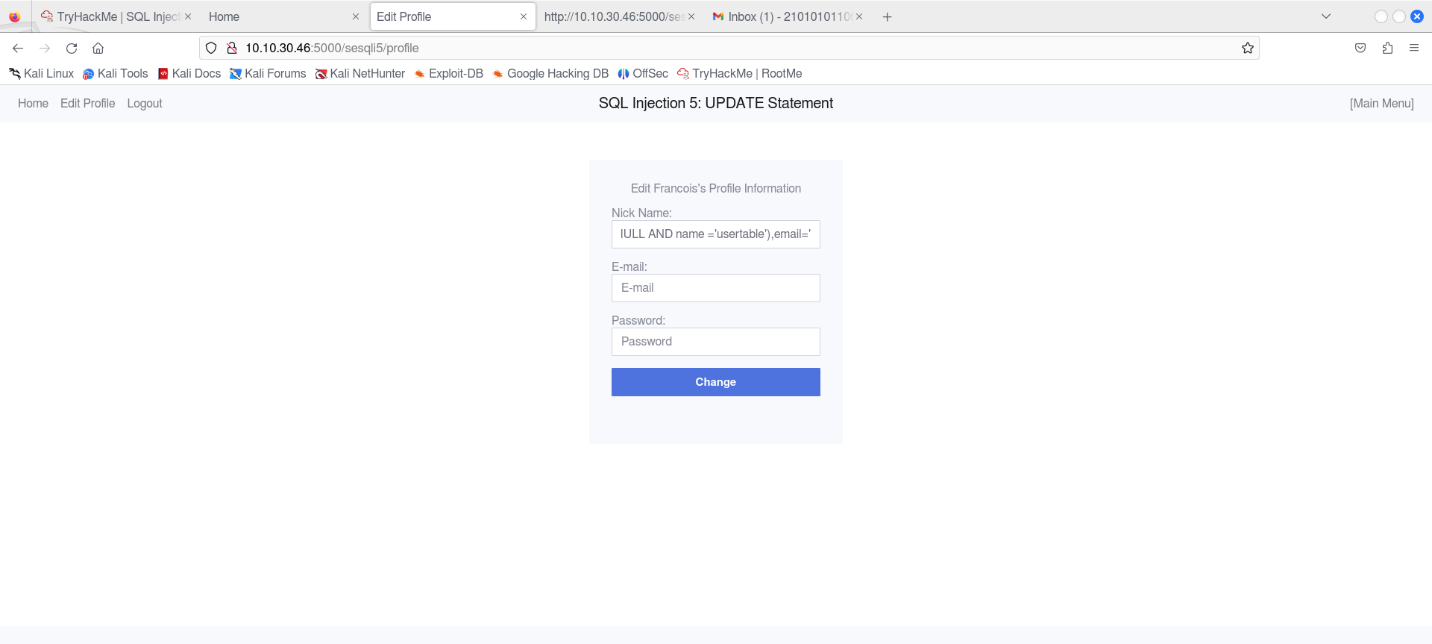
By injecting the code above, we can see that the table in the database is called "usertable" and “secrets”

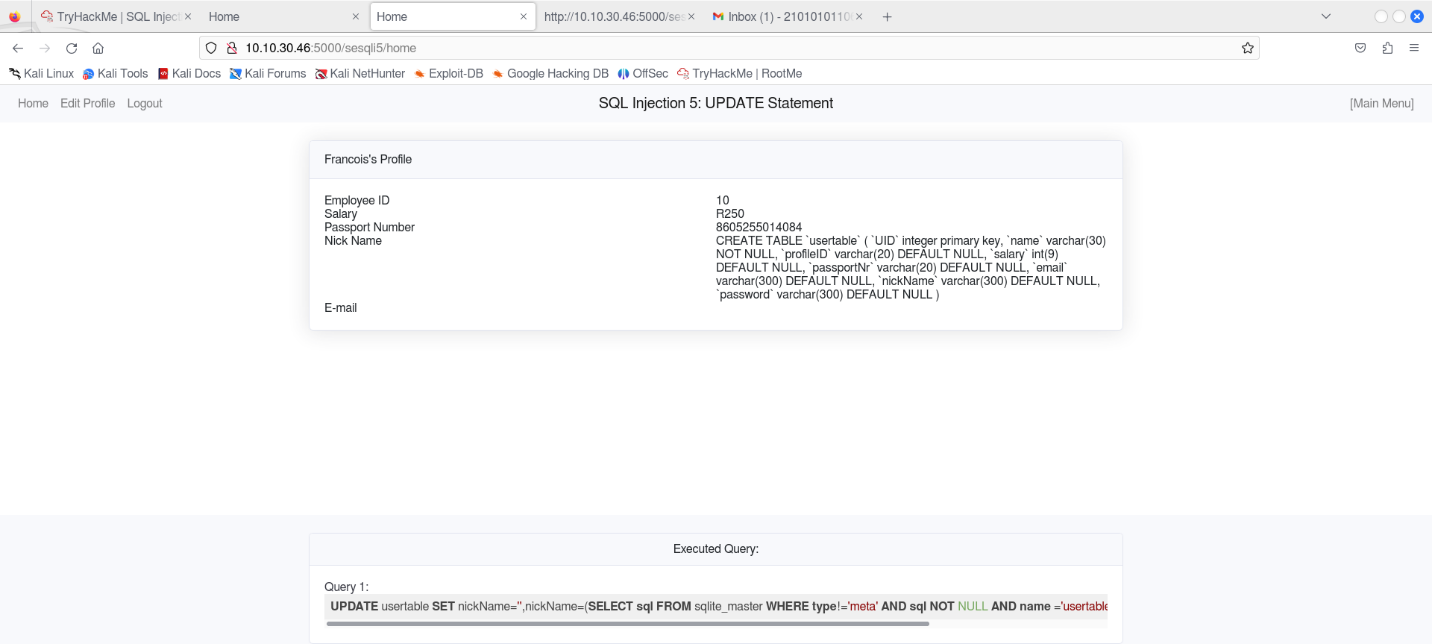




We can then continue by extract all the column names from the usertable:

*',nickName=(SELECT sql FROM sqlite\_master WHERE type!='meta' AND sql NOT NULL AND name ='usertable'),email='*

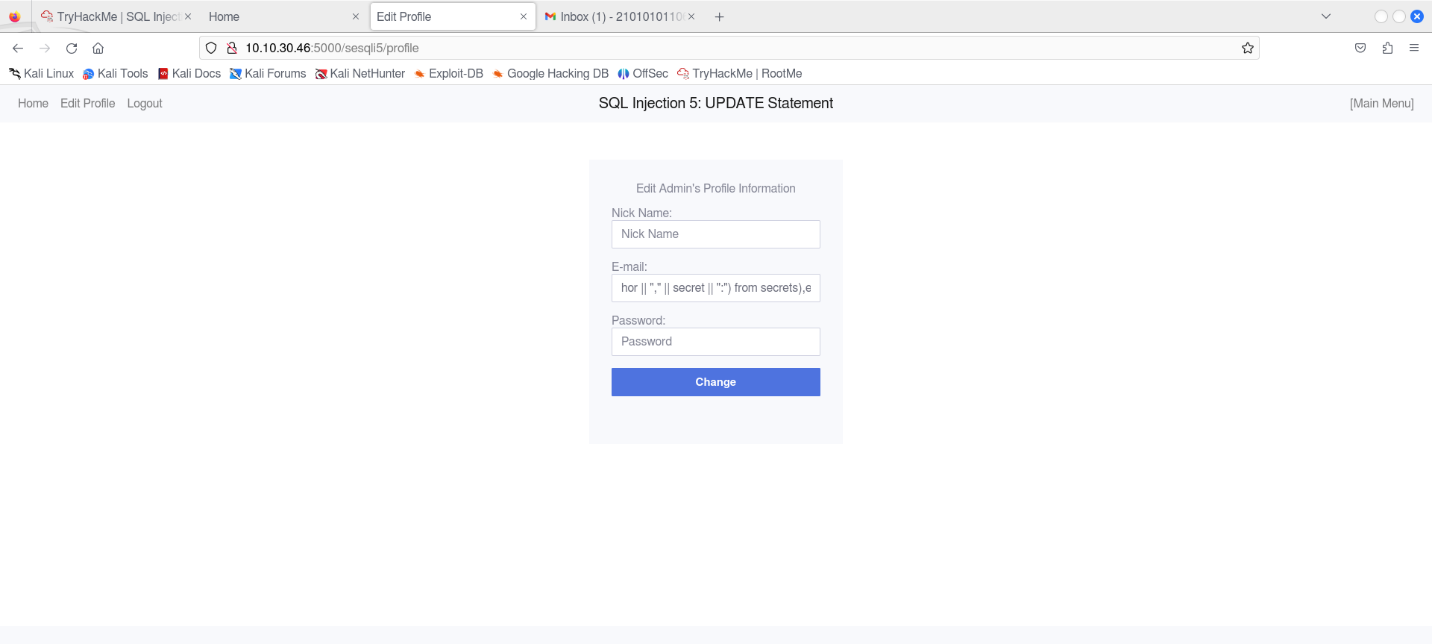


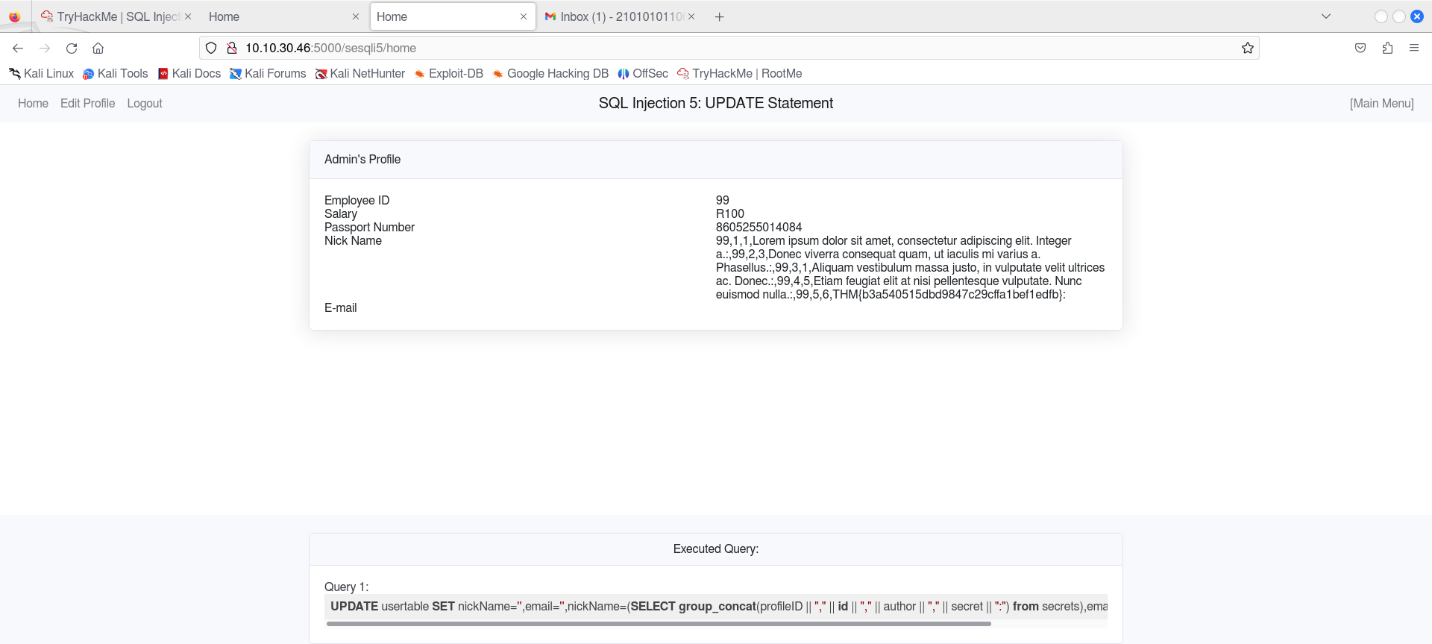


After having dumped the data from the database, we can see that the password is hashed. This means that we will need to identify the correct hash type used if we want to update the password for a user. Using a hash identifier such as hash-identifier, we can identify the hash as SHA256:

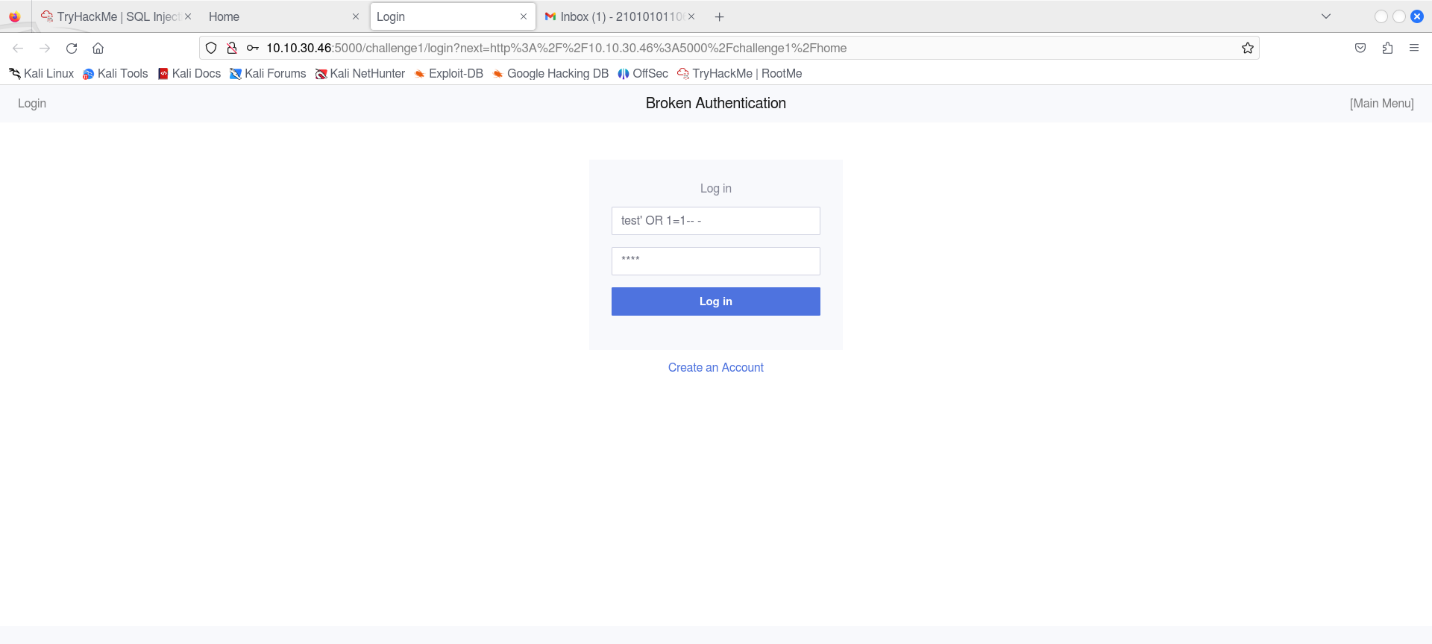
There are multiple ways of generating a sha256 hash. For example, we can use <https://gchq.github.io/CyberChef/>:



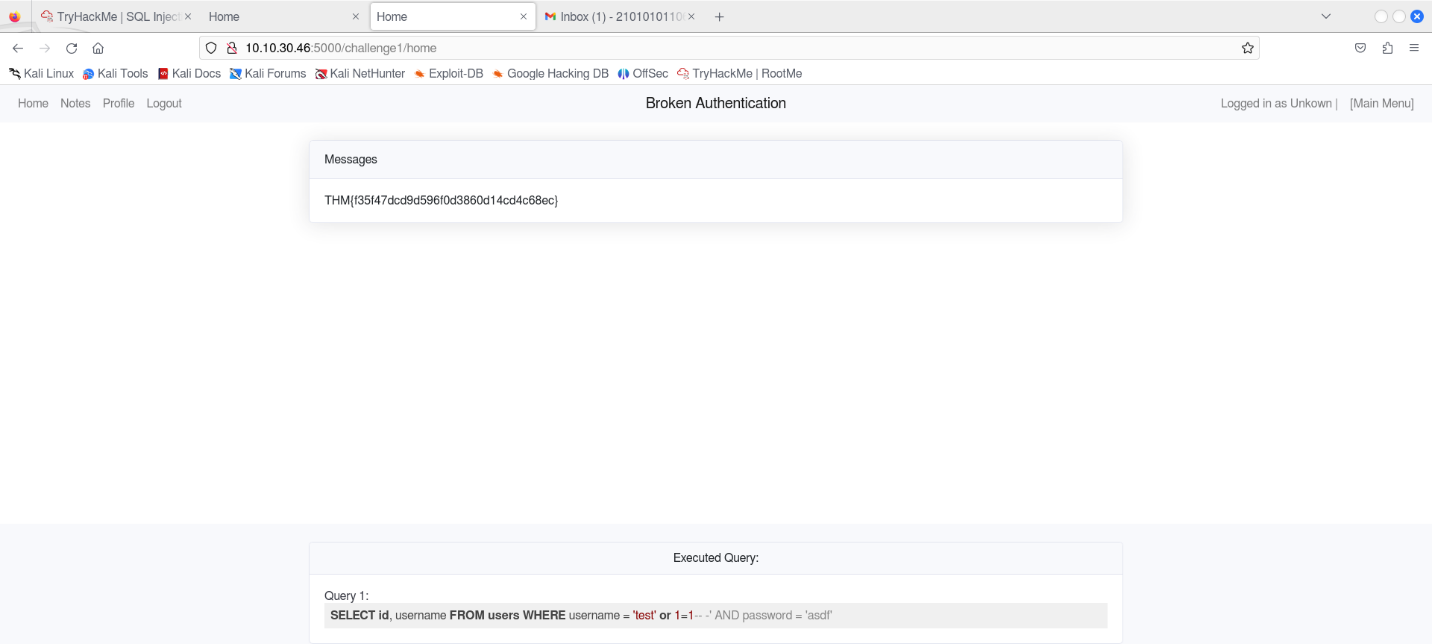




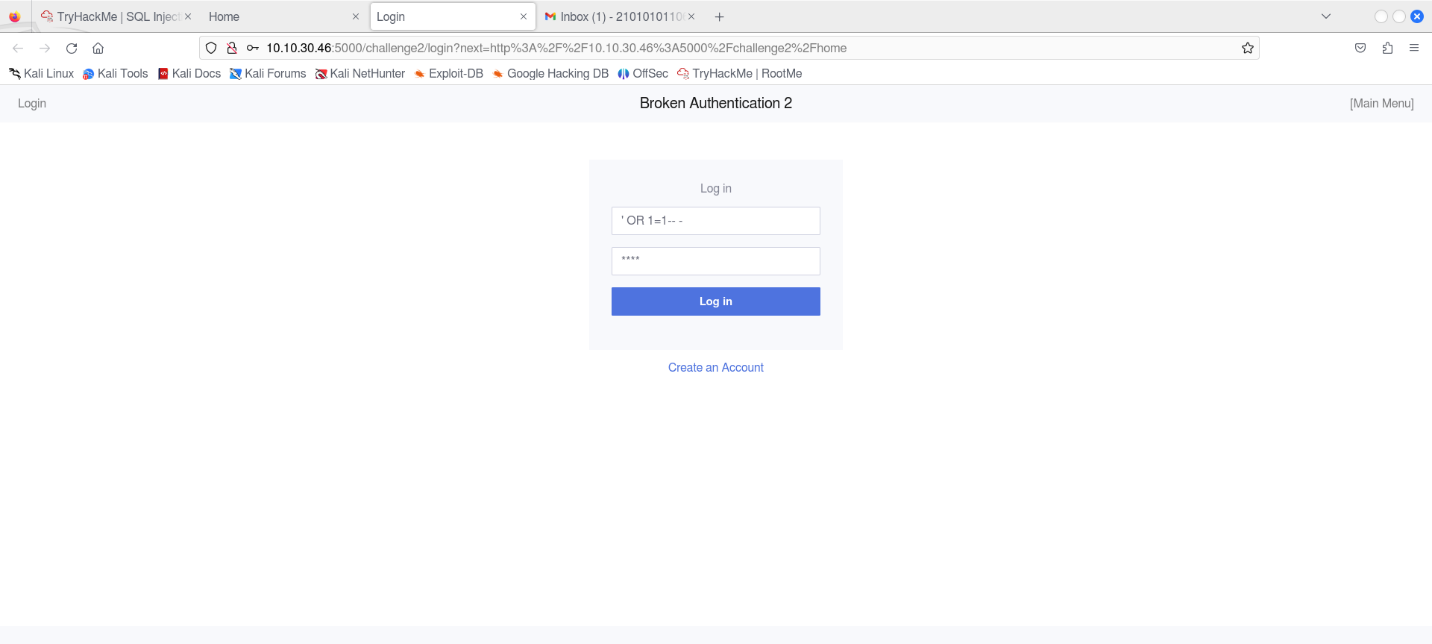
**Vulnerable Startup: Broken Authentication**



From the landing page on [http://10.10.30.46:5000](http://10.10.30.46:5000/" \t "_blank), go to Broken Authentication under Track: Vulnerable Startup (<http://10.10.30.46:5000/challenge1/>).



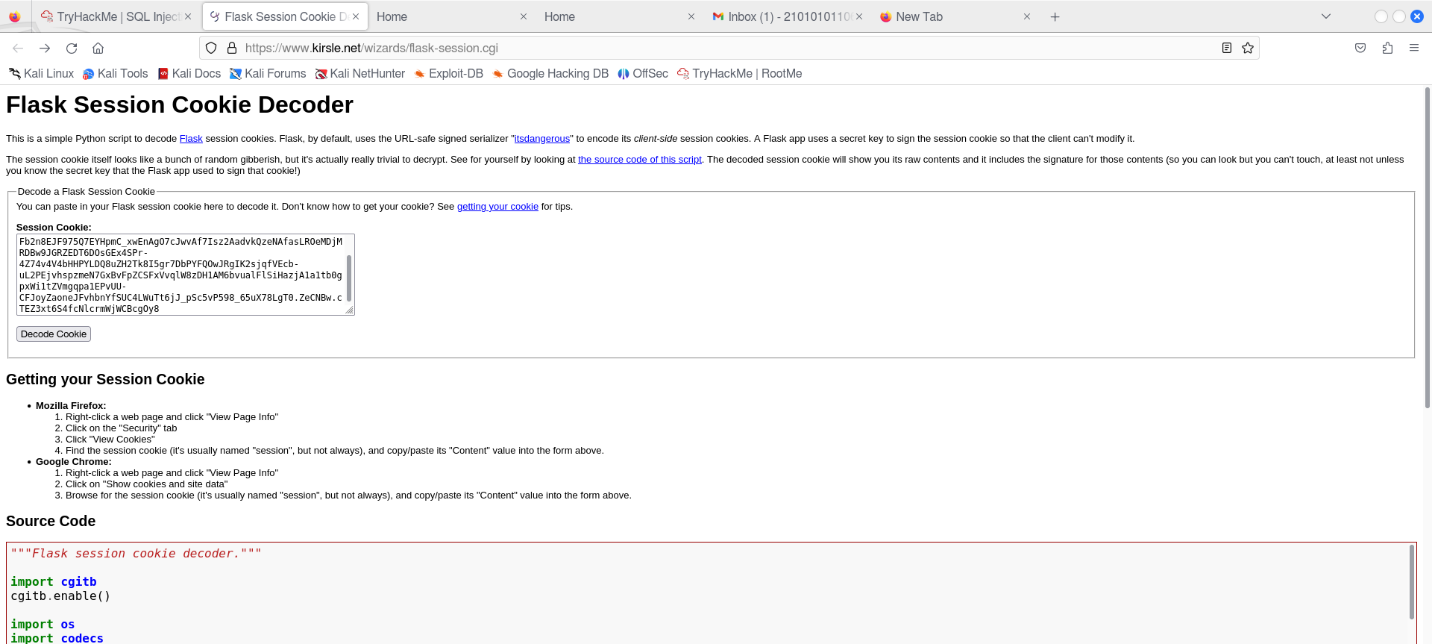
**Vulnerable Startup: Broken Authentication 2**

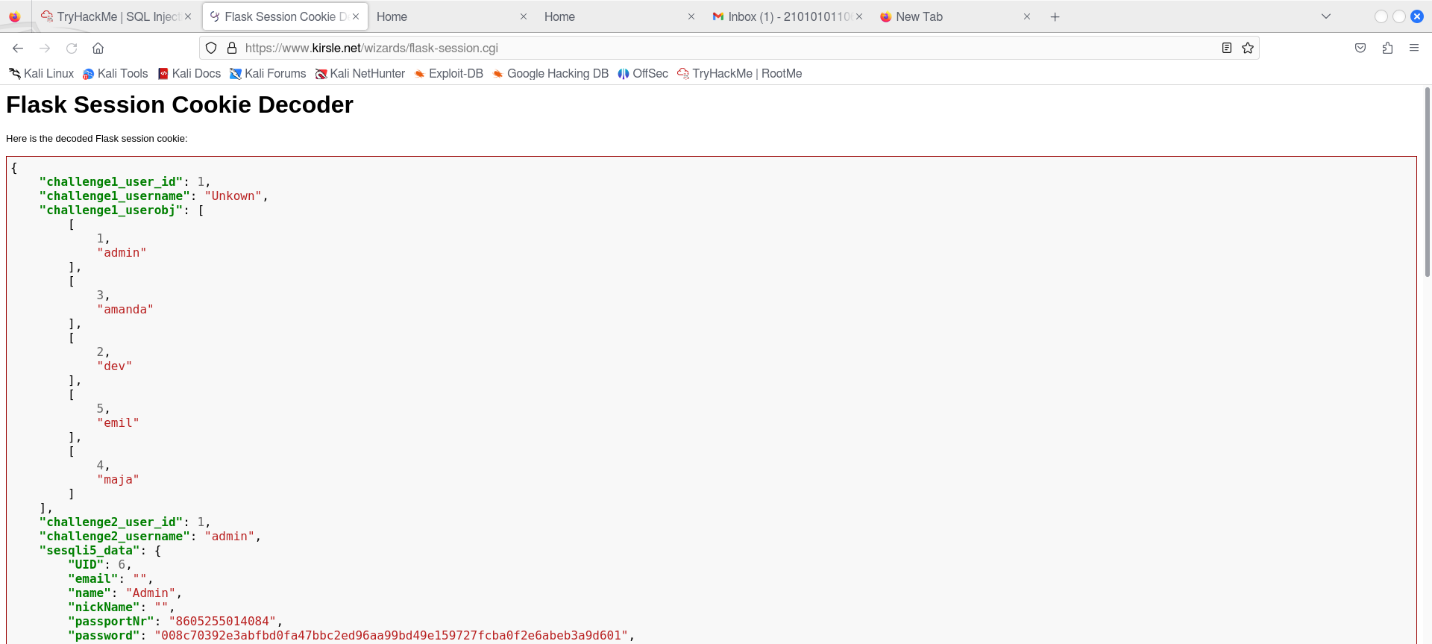


The login form is still vulnerable to SQL injection, and it is possible to bypass the login by using ' OR 1=1-- - as a username.

Before dumping all the passwords, we need to identify places where results from the login query is returned within the application. After logging in, the name of the currently logged-on user is displayed in the top right corner, so it might be possible to dump the data there.

Data from the query could also be stored in the session cookie. It is possible to extract the session cookie by opening developer tools in the browser, which can be done by pressing F12.





When logging in to the application, it executed the query below. From the SQL statement, we can see that it is retrieving two columns; id and username.

*SELECT id, username FROM users WHERE username = '" + username + "' AND password = '" + password + "'*

Without knowing the number of columns upfront, the attacker must first enumerate the number of columns by systematically injecting queries with different numbers of columns until it is successful. For example:

*1' UNION SELECT NULL-- -*

*1' UNION SELECT NULL, NULL-- -*

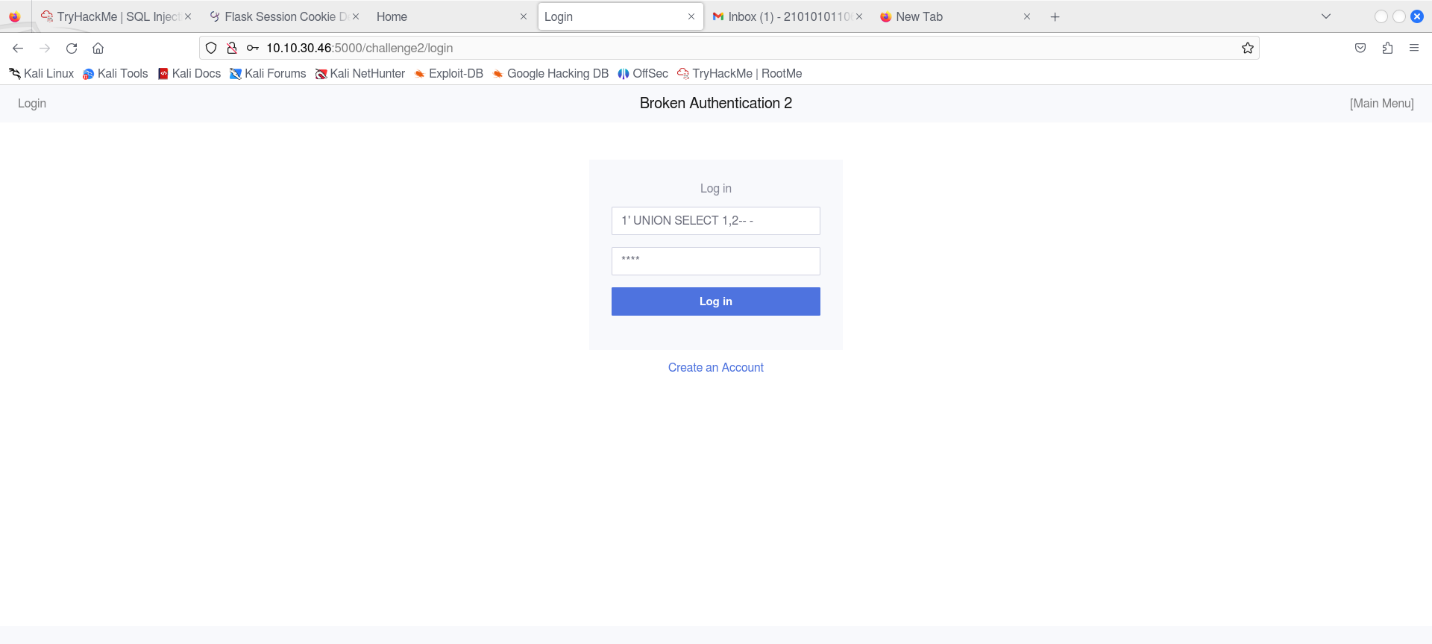
*1' UNION SELECT NULL, NULL, NULL-- -*

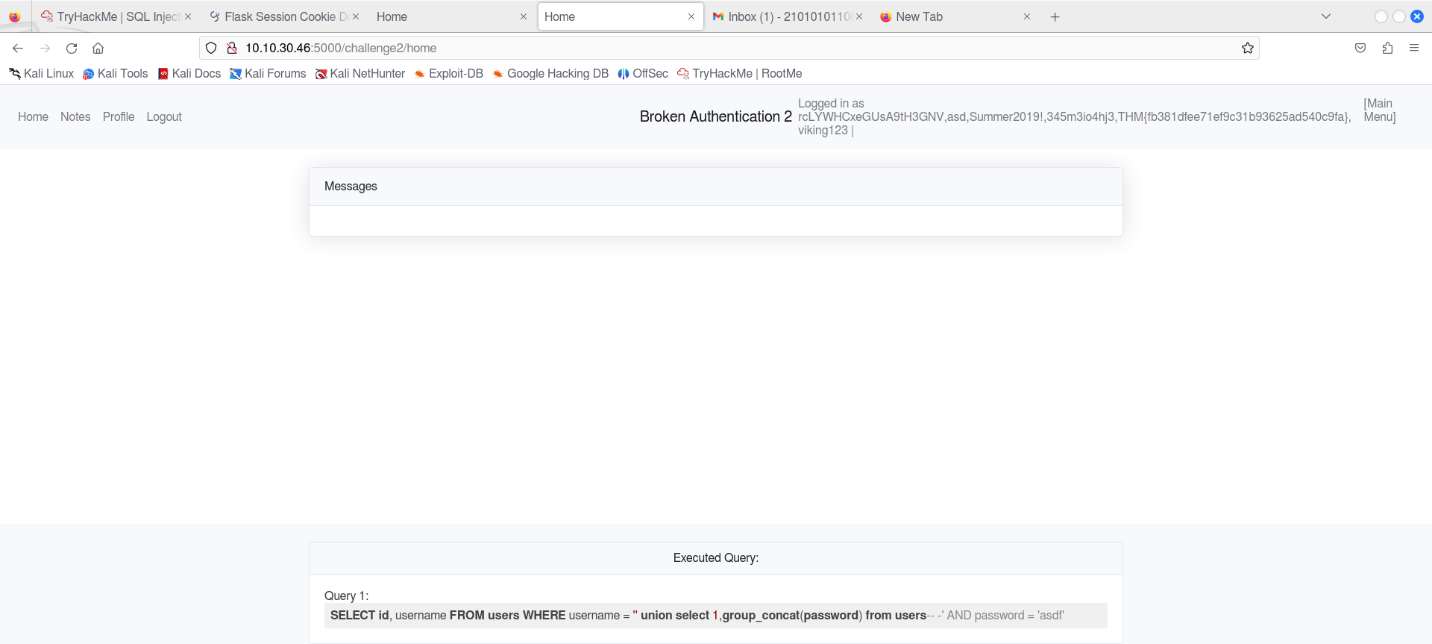
*By using ' UNION SELECT 1,2-- - as username, we match the number of columns in the original SQL query, and the application lets us in. After logging in, we can see that the username is replaced with the integer 2, which is what we used as column two in the injected query.*

By injecting the following code into the username field:

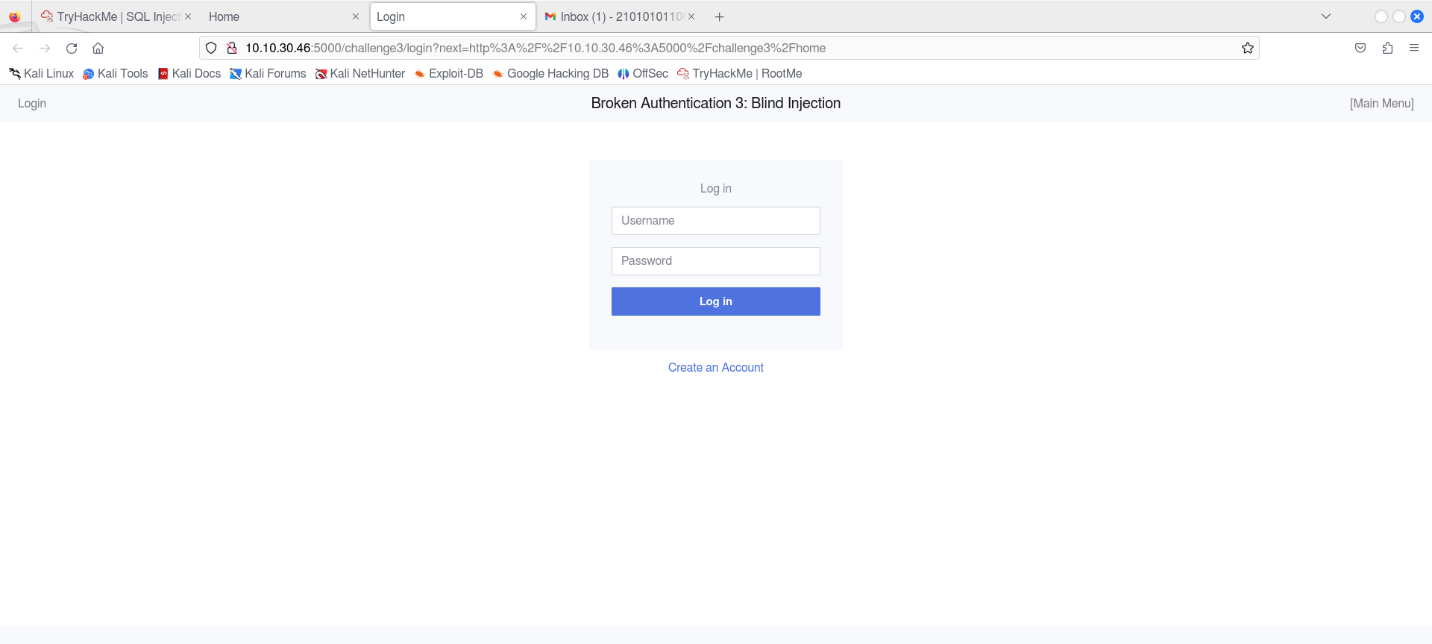
*' UNION SELECT 1,group\_concat(password) FROM users-- -*

All the passwords are dumped:

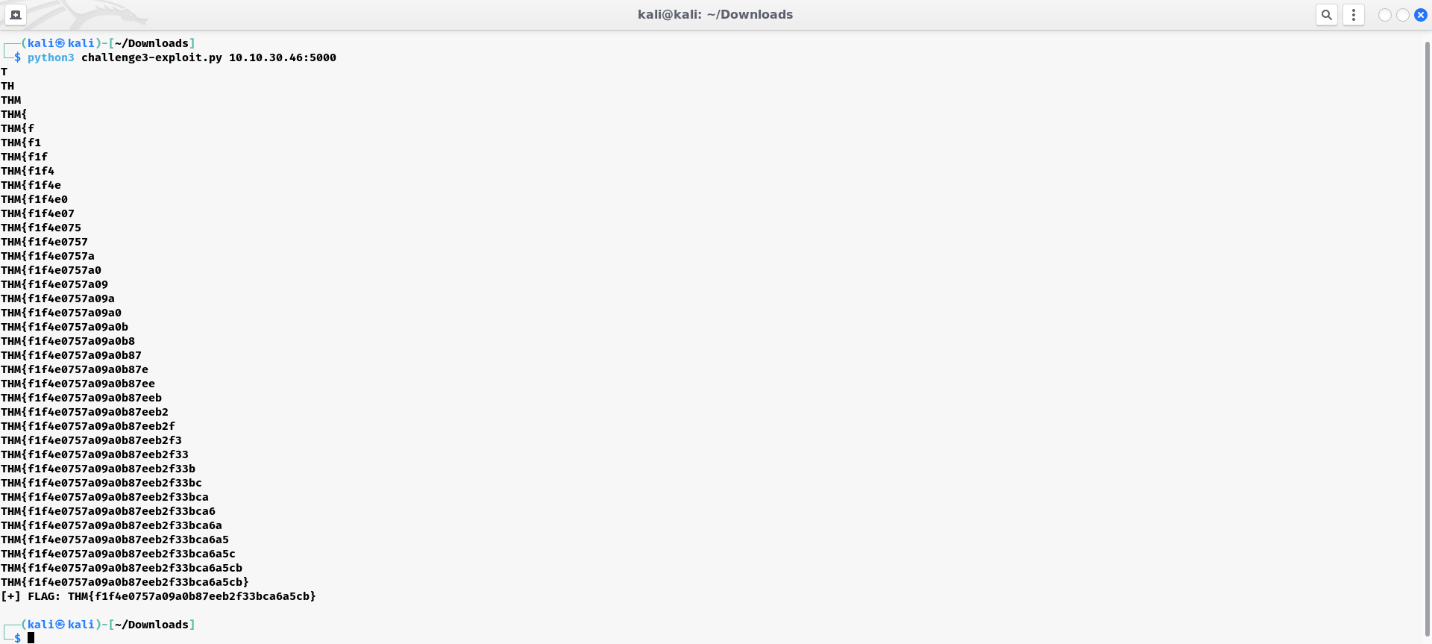




**Vulnerable Startup: Broken Authentication 3 (Blind Injection)**



An example script is provided inside the machine and can be view and downloaded by going to [http://MACHINE\_IP:5000/view/challenge3/challenge3-exploit.py](http://machine_ip:5000/view/challenge3/challenge3-exploit.py" \t "_blank); note that it will be necessary to change the password length with the password\_len variable. The length of the password can be found by asking the database. For example, in the query below, we ask the database if the length of the password equals 37:



An alternative way to solve this challenge is by using a tool such as sqlmap, which is an open source tool that automates the process of detecting and exploiting SQL injection flaws. The following command can be used to exploit the vulnerability with sqlmap:

*$ sqlmap -u http://MACHINE\_IP:5000/challenge3/login --data="username=admin&password=admin"*

*--level=5 --risk=3 --dbms=sqlite --technique=b --dump*