

1. Install MySQL and Python

You have a site that requires you to log in with your username and password to access the pages that have information about your account. When you log in, the credentials are passed to the database. If the username and password match what is in the database, you will be **authenticated** for a session. A session is a specified amount of time that an authenticated user will have access to specific pages and activities on the application.

At any time in the authentication process, a malicious user can **gain unauthorized access to the session**. As attacks get more sophisticated, it is more important for web applications safeguard against them.

Broken authentication can include automated attacks such as using a list in a **brute force** or **credential stuffing** attack. Brute force is done by trying as many username and password combinations as possible until getting the right one! When a malicious user has unauthorized access to a list of usernames and passwords, an attack called credential stuffing can be used to try all of the username and password combinations until this hacker authenticates.

How to Fix Broken Authentication?

- Implement multi factor authentication ...
 - Use an SSL certificate ...
 - Enforce strong passwords ...
 - Control session timeout ...
 - Rotate and invalidate session IDs ...
 - Don't put session IDs in the URLs ...
 - Don't store passwords in cleartext ...
 - Use breached password protection ..
-
- ## 2. Create secureusers table
- a. We are going to create secure users table just like the last time, but with a tiny change. We are going to introduce two new columns- one to store the One-Time-Password in a hashed format- and its key so that we can validate the OTP once the user enters it.
 - b. It is important to note that we should not store OTP in the plain text either- if we encrypt it, someone may still be able to decrypt them if they gain access to the key. Hashing is one way, and even we can't reverse engineer it.

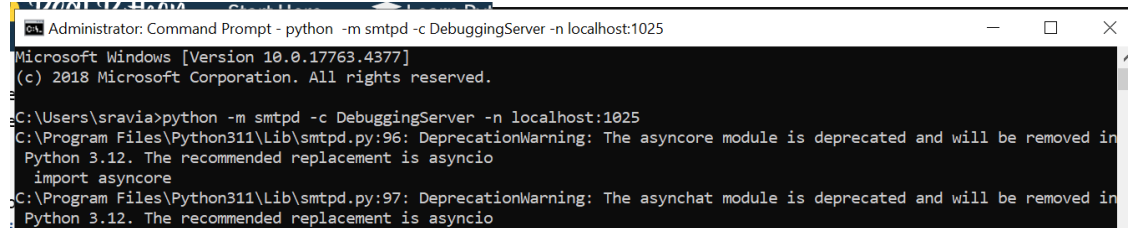
```
mysql> CREATE TABLE secureusers (
->  uname VARCHAR(256),
->  upass VARCHAR(256),
->  usecret VARCHAR(256),
->  usa1t VARCHAR(256),
->  ukey VARCHAR(256),
->  otp VARCHAR(256),
->  otp_sa1t VARCHAR(256)
-> );
Query OK, 0 rows affected (0.03 sec)
```

- c.
3. Open a command prompt, and run our local SMTP server
 - a. Simple Mail Transfer Protocol for DEV TEST ONLY. For production, we may use a real server like SendGrid or Office 365 or Gmail or MailChimp amongst others.
 - b. This will run a server locally on our machine
 - c. It will NOT send any real email
 - d. Instead, it will display email content on our command window itself
 - e. In production, we could replace it with a real email server like SendGrid or Gmail or Office 365
 - f. To open a new SMTP for dev/test, we will use:
 - i. `python -m smtpd -c DebuggingServer -n localhost:1025`
4. Open a new command prompt window and run following commands:
 - a. `Mkdir newapp`
 - b. `Cd newapp`
 - c. `Python -m venv myenv`
 - d. `Myenv\Scripts\activate`
 - e. `Pip install flask and mysql-connector-python` and other libraries
 - f. Git Clone the prepare repo- observe that this time we have 2 files- one called `twofactor.py` which defines a `send otp` function. This function will be imported in our `API.py`- and this is an example of how we can use modules to make our main Flask API lesser in code length and easier to debug. Now we can isolate errors of sending OTP from the API errors if they were to arise.
 - i. < Open a new terminal, cd to otp folder, and `python -m idlelib twofactor.py`>
 - ii. In this file, observe that we have a `sendotp` function using SMTP on port 1025 of localhost. We are forming a message before sending this to SMTP server. Keep in mind, that this dev-test server will display the emails on terminal, and not send them actually.
 - iii. In this file, observe how we have a `sendotp` function. This function is of the same name as our module `twofactor`. Py. So when we call this function, we have

to be careful on whether we are calling a local function, or if we are calling the module function (Line 41)

- iv. This function does not have a route- meaning we are not exposing it to internet, and only using it within our file

- g. DO NOT CLOSE THIS TERMINAL otherwise our SMTP server will not work



```
Administrator: Command Prompt - python -m smtpd -c DebuggingServer -n localhost:1025
Microsoft Windows [Version 10.0.17763.4377]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\srvavia>python -m smtpd -c DebuggingServer -n localhost:1025
C:\Program Files\Python311\Lib\smtpd.py:96: DeprecationWarning: The asyncore module is deprecated and will be removed in
Python 3.12. The recommended replacement is asyncio
  import asyncore
C:\Program Files\Python311\Lib\smtpd.py:97: DeprecationWarning: The asynchat module is deprecated and will be removed in
Python 3.12. The recommended replacement is asyncio
```

- h.