Bcrypt

Contents

Overview

Web application often require some sort of sensitive data storage, such as passwords and payment details. However, it is a security risk to store such data in a way that is easily visible. To solve this, we hash the sensitive information.

What is hashing?

Hashing is when we apply some cryptographic function to a string that converts it into fixed length string made up of seemingly random characters.

The properties that make hashing secure:

- It would take an unreasonably amount of time to attempt to reverse engineer a hashed string
- It is an injective function. That is, no two different strings will result in the same hash.

Using this, we can then simply compare hashes for logic, rather than the raw input.

Bcrypt

To use hashing in Flask, we use Flask-Bcrypt. A module that contains a class with hashing methods available to use. All we have to do is create the bcrypt object using our app:

```
from flask_bcrypt import Bcrypt

bcrypt = Bcrypt(app)
```

Bcrypt includes two useful methods that we will use:

- bcrypt.generate_password_hash('some string') - generates a hash.
- bcrypt.check_password_hash(hash,'string') - checks if the string when hashed equals the hash provided.

Note: the hash always comes first in the arguments for the check.

Tutorial

In this tutorial we will install Bcrypt and generate a hash for some string

Install Flask-Bcrypt:

```
pip3 install flask-bcrypt
```

Next lets create a flask application with a simple table. In a file named `app.py`:

```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt

app = Flask(__name__)

app.config['SQLALchemy_DATABASE_URI'] = "sqlite:///"

db = SQLAlchemy(app)

bcrypt = Bcrypt(app)
class Register(db.Model):
    name = db.Column(db.String(30), nullable=False, primary_key=True)
    password = db.Column(db.String(30), nullable=False)

db.create_all()

@app.route('/', methods=['GET','POST'])
def home():
    if request.form:
        person = Register(name=request.form.get("name"),
password=bcrypt.generate_password_hash(request.form.get("password")))
        db.session.add(person)
        db.session.commit()
    registrees = Register.query.all()
    return render_template("home.html", registrees=registrees)

if __name__ == '__main__':
    app.run(port=5000, debug=True, host='0.0.0.0')
```

Next, lets create a page with an input form, named 'home.html':

```
<html>
  <head>
    <title>Add person</title>
  </head>
  <body>
    <h1>Register</h1>
    <form method="POST" action="/">
      <input type="text" name="name">
      <input type="password" name="password">
      <input type="submit" value="register">
    </form>

    <h2>Registrees</h2>
    <table>
      {% for person in registrees %}
        <tr>
          <td>
            {{ person.name }}
          </td>
          <td>
            {{ person.password }}
          </td>
        </tr>
        {% endfor %}
    </table>
  </body>
</html>
```

Place `home.html` inside a folder named `templates`, and ensure that `templates` and `app.py` are in the same directory.

Next run the app:

```
python3 app.py
```

Navigate to `127.0.0.1:5000` if working locally. If working on a virtual machine then ensure port 5000 is open and navigate to `[external-ip]:5000` where `[external-ip]` is your virtual machine's external IP.

Create a mock user. When created, the password should appear hashed.

Exercises

Implement some form of password authentication that uses the `bcrypt.check_password_hash()` method.