FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Department of Computer Engineering

1. Course , Subject & Experiment Details

Academic Year	**2024-25**	**Estimated Time**	**03 - Hours**
Course & Semester	**T.E. (CMPN)- Sem VI**	**Subject Name & Code**	**CSS - (CSC602))**
Chapter No.	**03 – Mapped to CO-1**	**Chapter Title**	**Basics of Cryptography**

Practical No:	5
Title:	Implementation of Salt and Pepper password protection technique
Date of Performance:	27/04/2025
Date of Submission:	3/04/2025
Roll No:	9913
Name of the Student:	Mark Lopes

Evaluation:

Sr. No	Rubric	Grade
1	On time submission Or completion (2)	
2	Preparedness(2)	

3	**Skill (4)**	
4	**Output (2)**	

Signature of the Teacher:

Date:

Title: Implementation of Salt and Pepper password protection technique

Why is hashing used?

A hashing algorithm transforms a stream of data into a string of characters of fixed length. For example the hash of password is **5f4dcc3b5aa765d61d8327deb882cf99** But if output string is changed even slightly the hash value changes completely. The hash of password1 is **7c6a180b36896a0a8c02787eeafb0e4c**

Hashing is a one way encryption, usually used in passwords. When a password is registered it is hashed and on login the password is checked with the hash value. If the match is found login is successful.

Since the attacker doesn't have the actual value but the hash value, he cannot go backward and get the original password.

However if a commonly known password is used, it is easy to get the original password from the hash value using the rainbow table. A rainbow table is a database of commonly used passwords and their corresponding hash values. So hashing by itself it isn't enough to protect passwords.

So more secure techniques are used:

SALT:

Salt is a short string of random characters that is appended to the password before they are hashed. This helps in preventing rainbow table attacks.

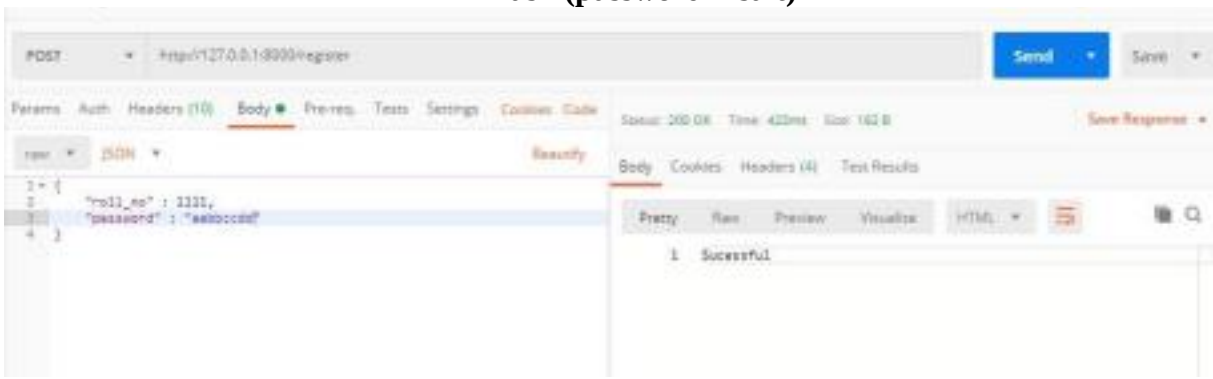
For e.g.: Password: **qwerty** may be in the rainbow table.

However **qwertyP#)!z** is not likely to exist in the database. Here **P#)!z** is the salt

used.

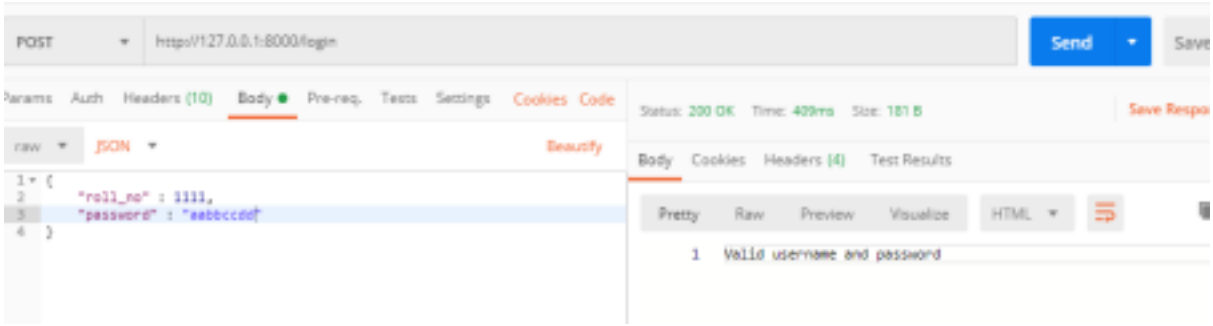
Salts are usually stored as plaintext and are added to the password, before storing in the database. The users are usually unaware of the salt.

Hash (password + salt)

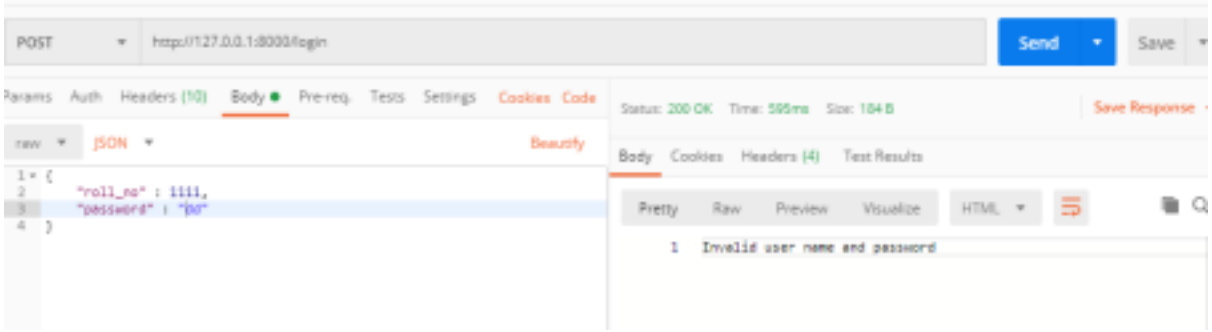


Register

Database



Successful login



Unsuccessful login

Advantage:

Salt aims at avoiding the issue with rainbow tables.

Disadvantage:
However the user can still obtain the password, if he gets the salt value and the location to add in the string.

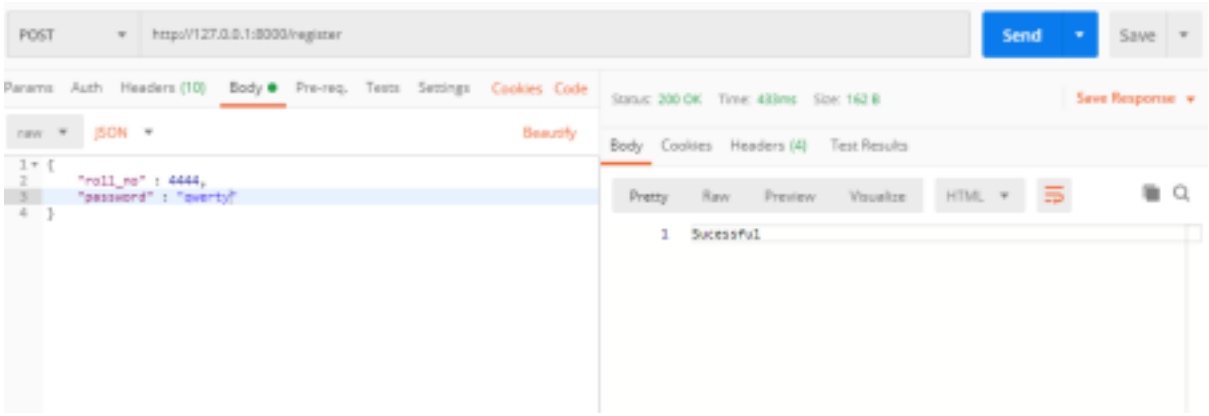
PEPPER:

Pepper is a short string or character appended at the end of the passwords. Peppers are random and different for each password.

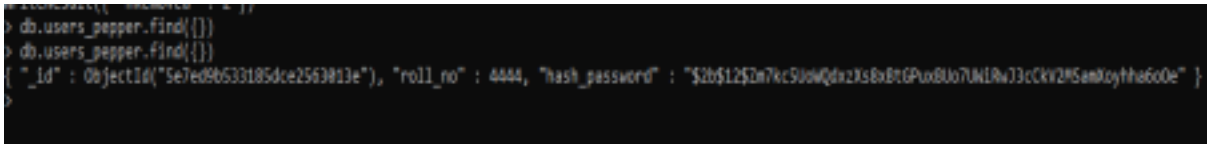
For e.g.: Password: **qwerty**

Pepper is letter '**e**'.

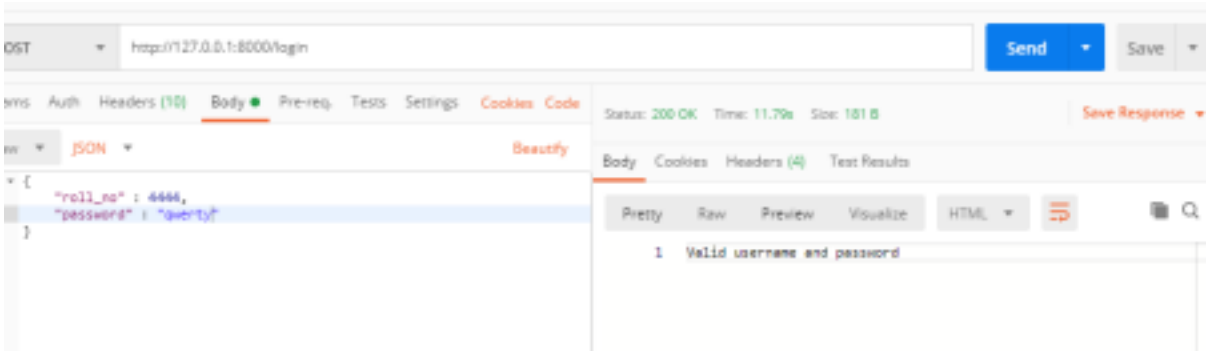
Hash (Password + Pepper)



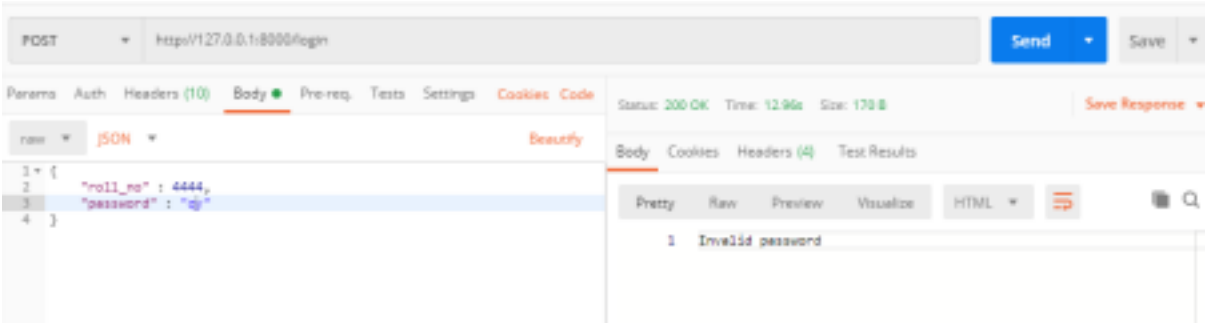
Register user



Database



Successful login



Unsuccessful login

Similar to salt the users are unaware of the Pepper. The pepper value is not stored. So when user enters a password the website cycles through all possible peppers until it matches with the hash.

Advantage:

This method is more secure since no one knows what the pepper is even the website, until it goes through all the possible options to find it.

Main.py

```
# app.py

from flask import Flask, render_template, request, redirect, url_for,
flash, session

from flask_sqlalchemy import SQLAlchemy

from werkzeug.security import generate_password_hash

import os

import hashlib

import secrets

import base64

from functools import wraps



# Initialize Flask app

app = Flask(__name__)

app.config['SECRET_KEY'] = secrets.token_hex(16)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False



# Initialize database

db = SQLAlchemy(app)



# Global pepper value (in production, this should be an environment
```

```
variable)

PEPPER = os.environ.get('PASSWORD_PEPPER', 'my_secure_pepper_value')



# User model

class User(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    username = db.Column(db.String(50), unique=True, nullable=False)

    email = db.Column(db.String(100), unique=True, nullable=False)

    password_hash = db.Column(db.String(256), nullable=False)

    salt = db.Column(db.String(64), nullable=False)



    def __repr__(self):

        return f'<User {self.username}>'





# Create database tables

with app.app_context():

    db.create_all()





# Custom password hashing with salt and pepper
```

```
def hash_password(password, salt=None):
    if salt is None:
        salt = secrets.token_hex(16)

    # Add the pepper to the password
    peppered_password = password + PEPPER

    # Hash the peppered password with the salt
    salted_pepper_pw = hashlib.pbkdf2_hmac(
        'sha256',
        peppered_password.encode('utf-8'),
        salt.encode('utf-8'),
        100000  # Number of iterations (higher is more secure but slower)
    )

    # Convert the hash to a storable string format
    password_hash = base64.b64encode(salted_pepper_pw).decode('utf-8')

    return password_hash, salt
```

```python

# Verify password
def verify_password(password, stored_hash, salt):
    calculated_hash, _ = hash_password(password, salt)
    return secrets.compare_digest(calculated_hash, stored_hash)




# Login required decorator
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_id' not in session:
            flash('Please log in to access this page', 'warning')
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function




# Routes
@app.route('/')
def index():
```

```
    return render_template('index.html')




@app.route('/register', methods=['GET', 'POST'])

def register():

    if request.method == 'POST':

        username = request.form.get('username')

        email = request.form.get('email')

        password = request.form.get('password')

        confirm_password = request.form.get('confirm_password')



        # Basic validation

        if not all([username, email, password, confirm_password]):

            flash('All fields are required', 'danger')

            return render_template('register.html')



        if password != confirm_password:

            flash('Passwords do not match', 'danger')

            return render_template('register.html')



        # Check if username or email already exists
```

```
        existing_user = User.query.filter(
            (User.username == username) | (User.email == email)
        ).first()

        if existing_user:
            flash('Username or email already exists', 'danger')
            return render_template('register.html')

        # Hash the password with salt and pepper
        password_hash, salt = hash_password(password)

        # Create new user
        new_user = User(
            username=username,
            email=email,
            password_hash=password_hash,
            salt=salt
        )

        db.session.add(new_user)
        db.session.commit()
```

```
        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        # Find user
        user = User.query.filter_by(username=username).first()

        if user and verify_password(password, user.password_hash,
user.salt):
            session['user_id'] = user.id
            session['username'] = user.username
            flash(f'Welcome back, {user.username}!', 'success')
```

```
            return redirect(url_for('dashboard'))



        flash('Invalid username or password', 'danger')



    return render_template('login.html')




@app.route('/dashboard')

@login_required

def dashboard():

    return render_template('dashboard.html')




@app.route('/logout')

def logout():

    session.clear()

    flash('You have been logged out', 'info')

    return redirect(url_for('index'))




@app.errorhandler(404)
```

```
def page_not_found(e):

    return render_template('404.html'), 404




if __name__ == '__main__':

    app.run(debug=True)
```