## Prevention & Patching

General prevention of timing-based vulnerabilities is difficult and depends on each web application's security issue(s). In our sample case, it suffices to do the database lookup based on username and password combined and only distinguish whether it was successful. The relevant code would then look like this:

Code: python

@app.route('/login', methods=['GET', 'POST'])  
def login():  
 if request.method == 'GET':  
 return render\_template('login.html')  
   
 username = request.form['username']  
 pw = request.form['password']  
 pw\_hash = bcrypt.hashpw(pw.encode(), salt)  
 user = User.query.filter\_by(username=username, password=pw\_hash).first()  
  
 if user:  
 session['logged\_in'] = True  
 session['user'] = user.username  
 return redirect(url\_for('index'))  
  
 return render\_template('index.html', message="Incorrect Details", type="danger")

Instead of querying the database only for the username, we do a combined lookup based on the username and the password hash.

However, in some instances, this is impossible. Consider a setting where the web application stores an individual password salt for each user. In that case, we can only compute the password hash after doing the database lookup based on the username. In these cases, we can eliminate the timing difference caused by the hashing of the password for valid users by hashing a dummy value if the username is invalid. In that case, the code would look similar to this:

Code: python

@app.route('/login', methods=['GET', 'POST'])  
def login():  
 if request.method == 'GET':  
 return render\_template('login.html')  
   
 username = request.form['username']  
 user = User.query.filter\_by(username=username).first()  
  
 if not user:  
 pw\_hash = bcrypt.hashpw(b'dummyvalue', salt)  
 return render\_template('index.html', message='Incorrect Details', type='danger')  
  
 pw = request.form['password']  
 pw\_hash = bcrypt.hashpw(pw.encode(), salt)  
  
 if pw\_hash == user.password:  
 session['logged\_in'] = True  
 session['user'] = user.username  
 return redirect(url\_for('index'))  
  
 return render\_template('index.html', message='Incorrect Details', type='danger')

Note that the web application hashes the value dummyvalue when the username is invalid. Thus, the bcrypt hash function is called whether the user is valid or invalid, resulting in no noticeable timing difference. However, this approach creates load on the server even for invalid usernames. Therefore, it is vital to implement proper rate-limiting on the login endpoint to eliminate the possibility of server overload and, subsequently, denial-of-service (DoS).