★ SY306 Project 2 Security Implementations Doc
★ Dr. Richards Section 3351
★ Team 6: Jacob Harrison, Jacques Henot, Kestrel Kuhne
★ Limit input size for the post to some reasonable length (in Python):
    - For this security measure we added a text limit for the number of characters posted to the message board in one message. The limit is 150 characters and if the message exceeds the character limit the server will truncate all characters after the 150th character and end the truncated message with a '...'. This allows for the user to continue the rest of their message that got truncated into a successive post. The code we used to accomplish this was: content = (content[:150] + '...') if len(content) > 150 else content
★ Do not store passwords in plain text
    - The passwords are hashed using SHA256 before they are submitted to the database. We also salted each hash so that even two of the same passwords would not result in the same salt & hash.
    - pDigest=hashlib.sha256((form.getvalue("Password")+form.getvalue("Username")).encode("utf-8")).hexdigest()
    - cursor.execute(query,(html_escape(form.getvalue("Username")),html_escape(form.getvalue("Name")),pDigest))
★ Duplicate all/any JS checks in Python
    - We included many JS checks within our website to include nothing being entered, the allowable length for each of the text fields (50 for name, 30 for username, 30 for password), special characters for any text field aren't allowed, and the password must be 6 characters long and contain a digit. If any of these requirements are not met, onblur() will show an error in our created div field that will notify the user what their error is. The user is not capable of submitting the form until all text fields are filled out correctly. This prevents users from using html and JS injection attacks during this process. We then validate that the username is not already in use and if it is return an error message and take the user back to signup. All these checks are then duplicated by the server in Python to not allow access until all requirements are met and not invalid special characters are submitted in the login, signup, and message boards.
★ Mitigate the risk of HTML and JavaScript injections by escaping < and >
    - All < and > characters are not allowed in our JS program which won't allow the user to submit if a text field contains those characters. Then in Python we also have the invalid character check with our

html_escape_table which contains these characters and replaces them. This mitigates the risk of HTML and JS injections.
    - Ex:
    - if (/[!@#$%^&*(),.?":{}|<>]/g.test(x) == true) {
    - document.getElementById('fl_name_errors').innerHTML='These characters are not allowed: !@#$%^&*(),.?":{}|<>';
    - return false;
    - }
    - html_escape_table = {
    - "&": "&",
    - "": """,
    - "'": "'",
    - ">": ">",
    - "<": "<",
    - }
    - def html_escape(text):
    - return "".join(html_escape_table.get(c,c) for c in text)
- ★ Use parameterized queries
    - All editing with the database is used with parameterized queries to avoid SQL injections. The variables are not directly within the statement and they are loaded separately within the query statement. We pre-compiled our SQL statements so that the only thing we needed to supply were the parameters (variables) that needed to be inserted into the statement. Our use of parameterized queries is a means of preventing SQL injection attacks.
- ★ Implement captcha to prevent Cross-site Request Forgery Attacks
    - We used a captcha to prevent cross-site request forgery attacks. We used this captcha on the login page because the user can't access the message board from signup and can only access the message board from login. A captcha is used to prevent CSRF attacks when the session is launched and prevents this attack because we don't allow access the message board until the correct symbols are entered. The randomized form value for the captcha make the automated submission impossible because the CSRF attack program running has no way of knowing what the correct value to be entered is. Therefore an automated submission is not allowed and the session won't be hijacked. We provide error checking to make sure the captcha is filled out correctly.
- ★ Implement secure session management to prevent session hijacking/riding.

    - Session are checked and no one is allowed direct access to the message board unless they have logged in from the login page first. The username entered must match the username session that gets obtained from the login request. We also provide continuous checks throughout the user's time on the message board to validate if the session is still valid and if not, the session will time out and the user will be redirected to the login page.
- ★ Use HTTP digest authentication to secure access to an admin only portion of your website.
    - We created a page called admin.html (which contains a link that directs the user to the message board) that the user is redirected to if they provide the correct username and password to login as admin. Before allowing the user to access admin.html, we implemented digest authentication (.htaccess and .htpasswd in a separate admin folder that also contains admin.html) for that page which requires an additional username and password that we specified below at the next bullet point. The admin only portion of the website is that admin is allowed to delete any message from any user.
- ★ **Admin Credentials for HTTP digest authentication**
    - **Username: admin**
    - **Password: acme420**
- ★ We had several security defense implemented in Project 1 such as the JS field checks along with the corresponding Python checks. We had escaped invalid characters that could lead to injections and also had secure session management to prevent hijacking.