Secure Developers
Anthony Simas, Jose Quintas, Mohammad Immam, Sara Marigomen
Due 3/23/20
CIS4930 - Mini Project 1 Report

Discuss how to initiate exploits, type of vulnerabilities, how the mitigations were applied, difference between password hashing & encrypting password and what salting and salt + peppering passwords enable (compare / contrast).

Vulnerable Version: http://34.66.103.83/AccountR/Login
Mitigated Version: http://34.66.103.83/AccountH/Login

SQL Injection Attack

- To initiate the SQL injection, the attacker registers a dummy account with the user name 'jose' and password 'jose''. On the page " /AccountR/Login", the attacker enters into the username field an SQL injection like "'jose' or 1=1) --" (anyusername or …) and also enters into the password field any password present in the database (for example, the password "jose"). This logs the attacker in and maliciously enables the attacker to see everyone's to do list in the database.
- In the hardened version of the site (AccountH), we used the Microsoft SQL dataset instead of entity framework and its raw sql in order to mitigate sql injections. This ensures validation of input for SQL and for other malicious intentional and unintentional data entries.

IDOR Attack

- On the DashboardR accessed through the login of AccountR, the user is directly referenced by the username in the url. This is an indirect object reference. Any attacker can guess a username and exploit the dashboard to see the content of other users without logging in.
- In the DashboardH page, the communication of the username is done internally through the body and not directly in the URL. The mitigated version of the web application also makes sure to use the login system more efficiently. The server remembers the appropriate username who logged in through the AccountH controller and only responses to requests from the logged in username which is included in all requests.

CSRF Attack

- Within the dashboard, only the user should be able to mark their tasks complete. However, since the vulnerable version of the application doesn't protect against CSRF attacks, the attacker is able to create a short file that manipulates the forms within the dashboard to mark the To-Do items complete. For example, one can create a static website with a form that sends a post request to the url.
- http://34.66.103.83/AccountR/CompleteTodo?index=5&username=admin&isComplete=True" and mark the task complete for mohammad. https://storage.googleapis.com/sdcismp1forms/index.html is a link to a separate web site that demonstrates a threat to our vulnerable version of the web app
- This vulnerability is mitigated using ASP.NET AntiForgery Token validation. This makes sure that the server is aware of a token and it responds only to requests which must contain the token

in the form. Without the needed token, the form won't take any adjustments to the inputs. The example above also demonstrates the same request to the mitigated version which leads to an error.

XSS Attack

- For this attack, we have simulated a scenario where an attacker gained access to the user account of 'jose' and inserted some malicious external scripts and HTML to the database for him. The AccountR controller does not recognize this vulnerability, executes the script, and renders the HTML.
- This was mitigated using request-validation 2.0. Input validation was done on the database so that any malicious attacks are not executed but rendered as text only.

Hashing VS Encryption

- Hashing is generating a number from the given password. Once the hash is done, it is impossible to trace back to the original text, however, one given input produces the same output.
- Encryption is the process of encoding simple text and other information that can be accessed by the sole authorized entity if it has a decryption key. It is basically translating the plain text according to a chosen or created dictionary for strings.

Salting VS Salting and Peppering

- Salting a password is achieved by adding random bits to the end or to the beginning of a password in order to make it difficult to guess, to break or to decrypt. The salt is usually stored within the database itself for each password.
- Peppering is adding a constant secret to the password after salt to make the encrypted password harder to decrypt. The pepper is usually not stored inside the database, instead it is usually stored in a separate, secured location. For example, the pepper can be stored within the application itself, away from the database.
- If the password is just salted and encrypted, one would be unable to easily decrypt the password without the salt. If, however, an attacker gains access to the database with the stored hashed passwords and the salts, the attacker would be able to decrypt the passwords. Salting and peppering adds an additional layer of protection versus just salting a password. For example, if a hacker manages to gain access to the database with users login credentials (such as stored hashed passwords and salts), these stored credentials would be rather useless. Decrypting the password without the pepper is rather hard to achieve. Thus, salting and peppering passwords is a rather simple way to provide an additional layer of security in comparison to only salting passwords.