ChaimTube Activities 3-7 Questions

Wyatt Tauber, Shannon McHale, Connor Shade, Mike Lanzafame

Activity 3:

- **Provide a link to the test cases you generated for this activity.**

Travis CI Build #111

- **How do you ensure that users that navigate to the protected pages cannot bypass authentication requirements?**

A session cookie is set with the user ID and user name. The cookie must exist and have these values in order for the user to be logged in. The application checks for these requirements upon attempting to access a protected page, and redirects to the login page if these conditions are not met.

Our application is implemented in Flask. We are using Flask's default "itsdangerous" serializer to encode session cookies, signed with a 128-bit secret key. Without knowing the secret key, the client can't modify the session cookie to impersonate another user.

- **How do you protect against session fixation?**

Our application assigns a random 128-bit session ID to the user when they sign in. If the session ID is modified or moved between users, the assigned session ID will not match the session ID provided by the user and the user will be logged out. This prevents session fixation and session hijacking.

- **How do you ensure that if your database gets stolen passwords aren't exposed?**

The passwords stored in our database are salted with a random 16-bit salt and then encrypted using SHA-256. The hashing protects against our users' passwords being exposed, while the salt helps break common rainbow-table attacks.

- **How do you prevent password brute force?**

Our application uses Flask-Limiter to limit the number of login attempts per minute to 10. This number would be lower if not for our automated tests.

- **How do you prevent username enumeration?**

The incorrect.html page does not specify whether a username or password exists, only that the combination does not exist.

- **What happens if your sessionID is predictable, how do you prevent that?**

If the session ID is predictable, an attacker can impersonate another user by obtaining their session ID, although in this case they would also need Flask's secret key that the application encodes cookies with. However, our session ID and our secret key are sufficiently unpredictable as they use Python's pseudorandom number generator.

Activity 4:

- **Provide a link to the test cases you generated for this activity.**

Travis CI Build #221

- **How do you prevent XSS in this step when displaying the username of the user who uploaded the video?**

Special characters that could be used in cross-site scripting are removed from the user display name prior to being displayed on the webpage. In addition, these special characters are also removed from the video name prior to being stored in the database and saved to the server.

- **How do you ensure that users can't delete videos that aren't their own?**

We perform a two-step check to verify the identity of a user, and the owner of a video, before allowing deletion. The first step is to verify that the session id of a user is a valid session id (thereby verifying authentication). The second step is to verify that the user id of the current user is the user id of the video owner (thereby verifying authorization).

Activity 5:

- **Provide a link to the test cases you generated for this activity.**

Travis CI Build #221

- **How would you fix your code so that these issues were no longer present?**

  - Create a variable instead of having the cursor.execute raw sql. This can be seen in our original code. Create a variable for the username so it is not put in the middle of two tick marks. That is one of the things that makes this code truly vulnerable. Revert back to the 'incorrect.html' page that gives the attacker no indication of if their username or password is incorrect. Do not send an error message to the wrongpassword.html page

- **What are the limitations, if any that, of the SQL Injection issues you've included?**
  - A malicious hacker would not be able to directly log in using OR '1'='1 . For blind injection, the hacker will not be able to see if their username or password is incorrect. Because incorrect.html is in effect. Pymysql is the main limitation. This Python3 library does not allow a user to submit multiple lines of SQL injection. The Python2 dictionary MySQL allowed this vulnerability to occur.

Activity 6:

- **Provide a link to the test cases you generated for this activity.**

Travis CI Build #226

- **How would you fix your code so that this issue is no longer present?**

A simple fix to mitigate the SSRF vulnerability would be to perform additional checks on the file obtained via URL in order to ensure that the file is a .mp4 file and not an arbitrary file or command.

- **How does your test demonstrate SSRF as opposed to just accessing any old endpoint.**

A server side request forgery is any action that the client can cause the server to take on its behalf that otherwise would be denied if the client sent the request itself. As the client cannot access chaimtube.local/etc/passwd, a file handler was introduced that causes the server to request any file in its /etc folder if requested via the /file webpage. Requesting chaimtube.local/file/passwd causes the server to request chaimtube.local/etc/passwd, and since the server is allowed to do this, it returns the file to the client.

Activity 7:

- **Provide a link to the test cases you generated for this activity.**

Travis CI Build #221

- **How would you fix your code so that this issue is no longer present?**

This issue was introduced for the purposes of meeting this activity. The issue would be resolved through removing the command parameter from the /addsuser endpoint.

One potential reason for why this parameter would exist is so that a previous developer could have a default command run whenever a new user was added (possibly to send a Slack message, possibly to update logs, possibly to ring a bell - the possibilities are endless) but that our team has "not removed it yet for some reason", creating the vulnerability.