Application Security Assignment 2

By:

Name: Jeet Kamdar
netid: jsk757

Name: Nikhil Narasimha Prasad
netid: nnp267

Web Application Vulnerabilities and Fixes (mysite)

The web app provides security against some common vulnerabilities like:

1. Cross Site Scripting (XSS) attacks
2. SQL Injection
3. Cross site request forgery (CSRF)
4. Unrestricted file uploads

In addition, it provides session security. We have used Django for this assignment and it provides the following security features:

Cross Site Scripting

XSS attacks allow a user to inject some malicious scripts into the browsers of other users. This is achieved by storing a script in the database, and when it is retrieved and displayed to some other user, or by getting them to click on a link that will cause these scripts to be executed on the user's browser.

Django templates provides protections against these XSS attacks.

SQL Injection

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.

Django is protected against such attacks as the queries are constructed using query parameterization. The query's SQL code is defined separately from the parameters. Since parameters are provided by the user, they are unsafe as they can escape the underlying database driver.

Cross site request forgery (CSRF)

CSRF attacks allow a malicious user to execute actions using the credentials of another user without the knowledge or consent of the targeted user.

Django has built-in protection for this attack. CSRF protection works by checking for a secret in each POST request. This ensures that a malicious user cannot simply “replay” a form POST to your website and have another logged in user unwittingly submit that form. The malicious user would have to know the secret, which is user specific (using a cookie).

Unrestricted File Uploads

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step. The application prevents the user by uploading a file that is not a picture file, or a corrupted file posing as a picture file. This is done by checking whether the file is can be opened or not by the Pillow (PIL – Python Imaging Library) library, within an error handling block (using try-except). If no error is caught, the user’s image will be successfully uploaded and processed by the application. If there is an error, it is an indication that the uploaded file may be erroneous or corrupted. The user will be redirected to a separate page, and the file will be subsequently deleted from the application. This error handling block is implemented in the *views.py* file.

Session Security

Broken authentication and session management cause several security issues, all of them having to do with maintaining the identity of a user. If authentication credentials and session identifiers are not protected, an attacker can hijack an active session and assume the identity of a user.

It stores data on the server side and abstracts the sending and receiving of cookies which contain a session ID. We implemented sessions using the middleware 'django.contrib.sessions.middleware.SessionMiddleware' in the *settings.py* file.