Securing Web Applications

Assessment 3 for Secure Programming & Scripting

Zalán Tóth - 20102768
Computer Forensics and Security Year 2

SETU Waterford
School of Science and Computing
Department of Computing and Mathematics

Waterford City, Ireland
14/04/2024

Table of Contents

1. Part

A. HTTP POST

I wrote a very simple code, that uses the POST HTTP request to process data. The website takes in a name, and after submitting that name, it prints it out.

B. HTTP GET

This program is almost the same as the previous code from above. It changes the POST parts to GET. Also, it checks if the name is not null so the program doesn't get stuck at the submitted part.

Advantages & Disadvantages

When using GET, the link with the data can be bookmarked and shared, as the link includes the data in the query string. However, when dealing with sensitive data, we don't wanna expose that data in the link/query. When we're using POST, it doesn't expose the data in the link, so it's more suitable to handle sensitive data. It's also better to handle large amounts of data, but on the other hand, you cannot bookmark or share the link with the data.

C. XSS

Cross-site scripting (XSS) vulnerabilities occur when an application allows the attacker to inject executable scripts into the web content viewed by other users. My previous code is vulnerable to that kind of attack. So for example, if I submit the name "<script>alert('Hello')</script>", it'll execute the script as it injects this code into the paragraph.
The 1c.php code fixes that issue, so the attacker is not able to inject scripts. The only validation added was the htmlspecialchars. This function converts special characters to their HTML-encoded equivalents and protects the code from XSS.

2. Part

A. DB query

The login.html provides the form that needs to be submitted, then the login.php checks if the login credentials are correct, and tells you if the login is successful.

B. SQLi

The code in part A is vulnerable to SQL injections.
Performing SQLi:

01. If I log in with the following username:
    `admin' UNION SELECT * FROM users WHERE '1' = '1`
    That will inject the SQL request into the query which will make it execute. That way the web application will show a successful login because of the way the code was written. It can also reveal sensitive data as it extracts the users from the database.
02. If I log in with:
    `admin' --`
    That will bypass the password check and will allow me to log in without a password (although the page won't show anything). The part `--` makes the rest of the SQL code in that line a comment.
03. We could also inject drop tables or database queries and destroy the data if the user accessing the database has permissions to do so.

The code provided with this part is not vulnerable to SQLi, as it has validation against it.

3. Part

The following shows very basic input protections in HTML:

A. Maxlength

My example code demonstrates a limit on user name input in 10 characters, however the attacker can craft a custom HTTP request manually to bypass that limitation. As the limit is set on the client side, it's easy to go around it.

B. Readonly

This attribute keeps the field's value fixed and is not changeable by the user. However, the attacker can alter the HTML or the HTTP request and can bypass this easily. I provided a very basic demonstration with an example country field, and how it looks. I didn't integrate that into the database or the server side. It's only a client-side demonstration.

4. Part

A. PHP validation

The attached code showcases the usage of empty(), preg_match(), filter_input() and htmlspecialchars().

- The student ID is validated with preg_match() (so regular expressions).
- Filename is validated with preg_match().
- DoB is validated with preg_match() and strtotime() to verify the date.
- Email is validated with filter_input().
- The username is validated with htmlspecialchars().
- empty() is used several times in some if conditions.

The code includes client-side validation as well, but if we remove them, the server-side will still validate it, and if there is a mismatch it'll list out the errors from the errors array. It can be tested with the 2nd form, which doesn't provide client-side validation.

5. Part

A. Cookies

The attached code showcases a simple cookie detection and creation. If the cookie doesn't exist yet in the browser, it creates a new one. If the cookie exists already, it prints out the data of the cookie. For security, expiry can be used on the cookie. Also, cookies are observable easily, which means storing sensitive data in plaintext using cookies is not secure.

6. Part

A. Session variables

The attached codes are the following:

Start & set session

This code creates a session and assigns the session variables username and role. Then it prints out that data and the session ID.

Read session

This code continues the session and prints out the session variable values if the session is set.

Destroy session

This code continues the session and then unsets the session variables and then destroys the session.

B. Session Hijacking

Session hijacking is stealing or guessing a user's session ID, to gain unauthorized access to that session. The hijacking can happen as the following:

01. The user starts a session
02. Attacked obtains the session ID of the user
03. The attacker uses that session ID to access the site that is meant for the user.

To do that we can use the Burp Suite tool. This tool allows us to capture traffic, identify sessions, and clone them.