

Secure Programming

Assignment 2: Vulnerabilities

Lecturer: Stephen O'Shaughnessy



Digital Forensics and Cyber Security - TU765

Student: Robert Kacso (B00123508)

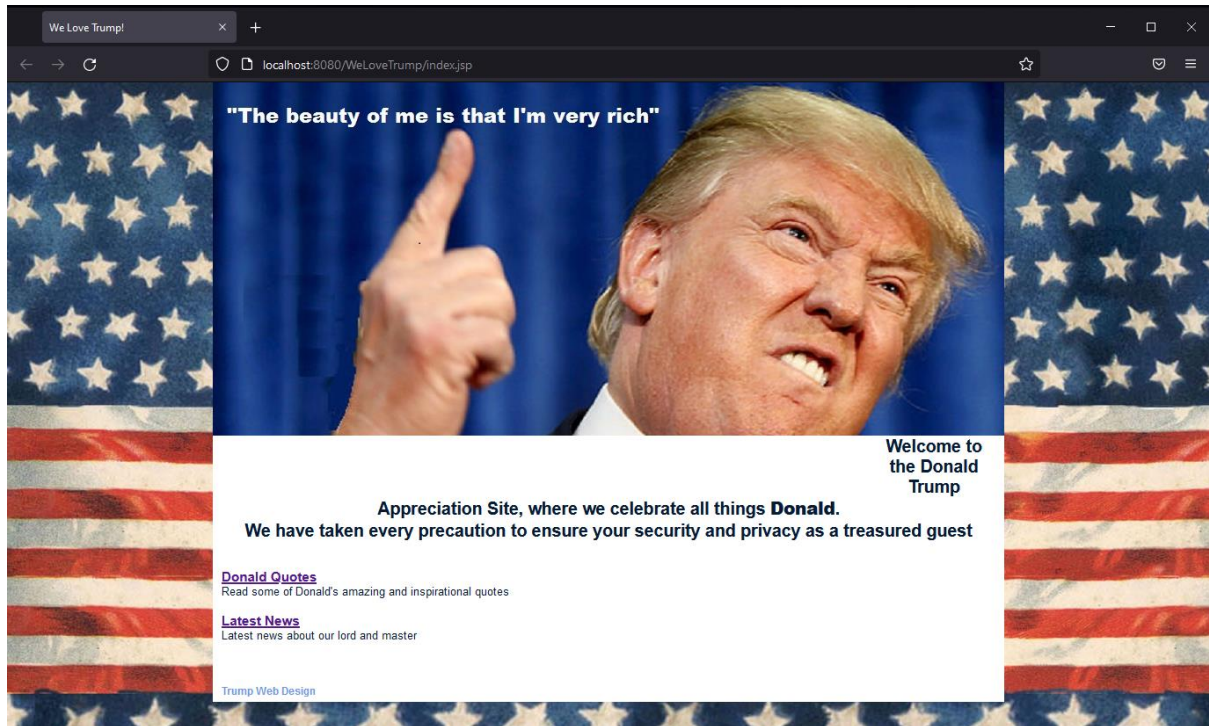
Date: 12th December 2021

Contents

CSS modification.....	2
Plain Text Password	3
Fixing Plain Text Password.....	4
SQL injections	4
Fixing SQL injections.....	7
Cross-site scripting (XSS)	7
Fixing Cross-Site Scripting Issues	10
IDOR (Insecure Direct Object References).....	11
Fixing IDOR	13
Security Misconfiguration	14
Fixing Security Misconfigurations.....	16
Unvalidated URL's	17
Fixing Unvalidated URL's.....	18
URL Modification.....	18
Fixing the URL Modification vulnerability	20
CSRF (Cross-Site Request Forgery).....	20
Fixing CSRF	21

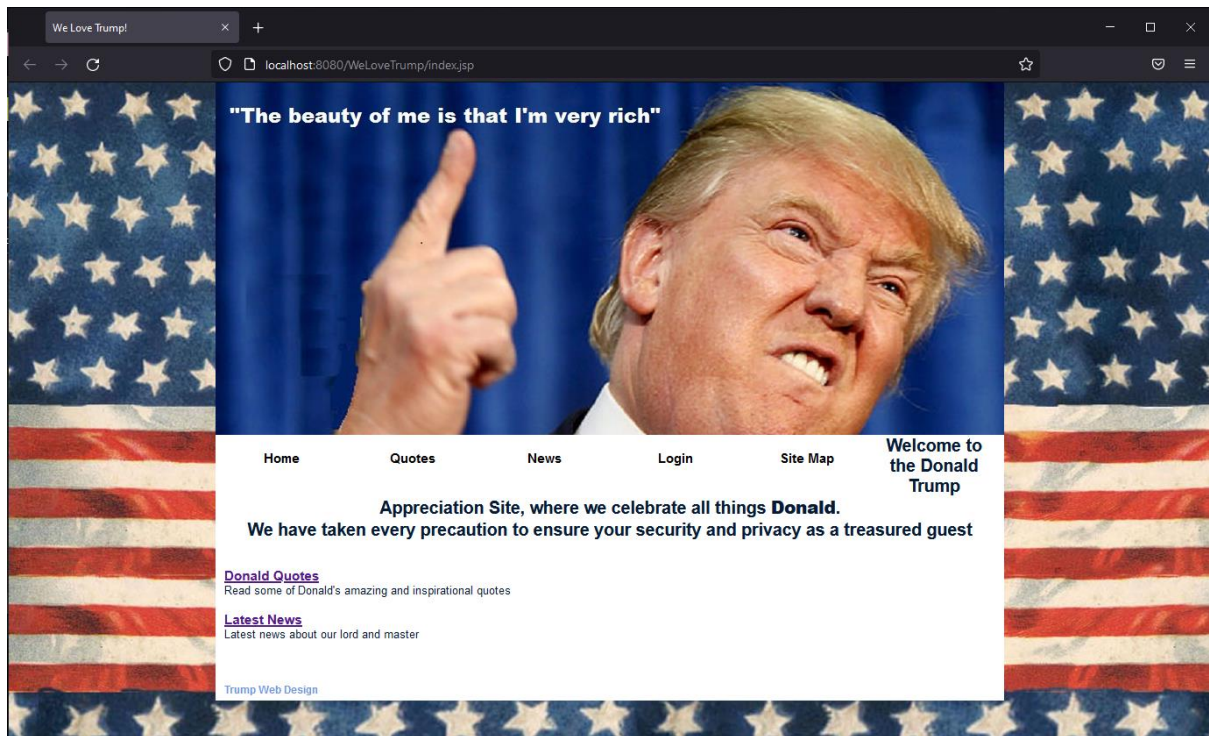
CSS modification

First thing after loading the website, I noticed that the menu bar was not visible only when hovering over it, so I modified the anchors in the menu bar to be the same colour as the rest of the text, so it'll be visible.



```
130
131 #menu a {
132     float: left;
133     width: 150px;
134     height: 40px;
135     display: block;
136     text-align: center;
137     text-decoration: none;
138     color: #ffffff;
139     font-weight: bold;
140     padding-top: 17px;
141     font-size: 15px;
142 }
143
```

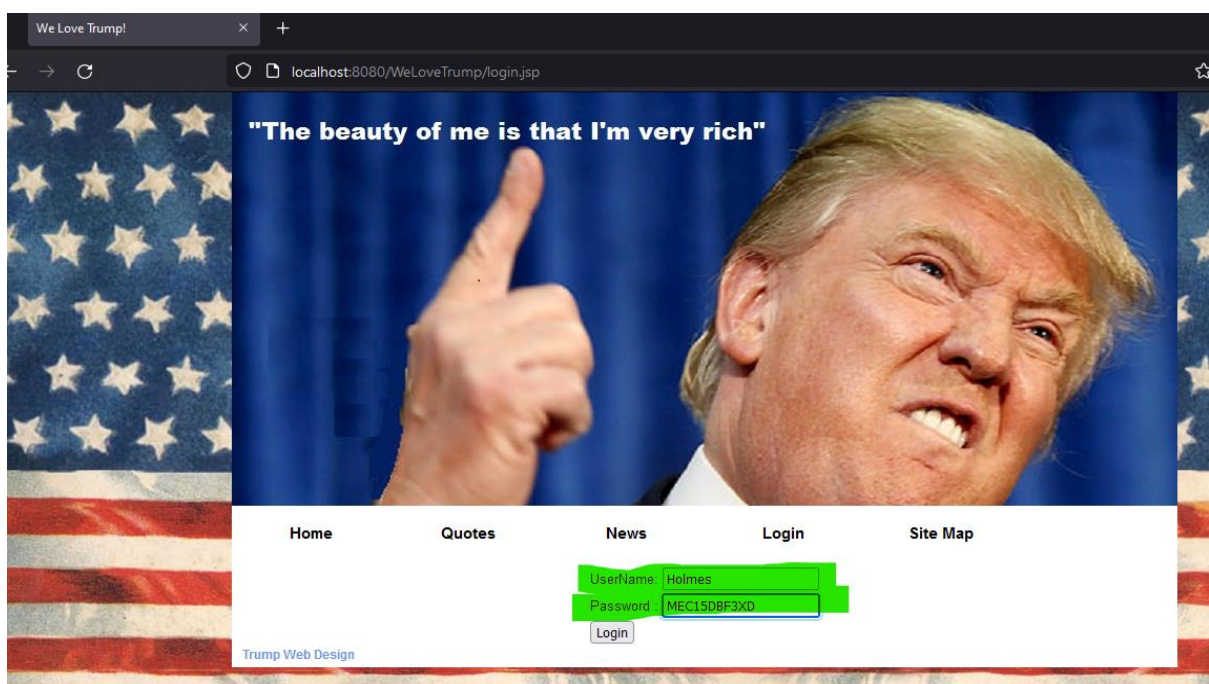
```
130
131 #menu a {
132     float: left;
133     width: 150px;
134     height: 40px;
135     display: block;
136     text-align: center;
137     text-decoration: none;
138     color: black;
139     font-weight: bold;
140     padding-top: 17px;
141     font-size: 15px;
142 }
```



Plain Text Password

After changing the CSS, I navigated to the login page and used the first user from the database to log in and to make sure that the webpage is communicating with the database.

As I typed in the Username and Password, I noticed that the password is shown in plain text, which is a vulnerability because anyone around can see it in plain text, instead of being hidden. It is also referred to as Shoulder Surfing.



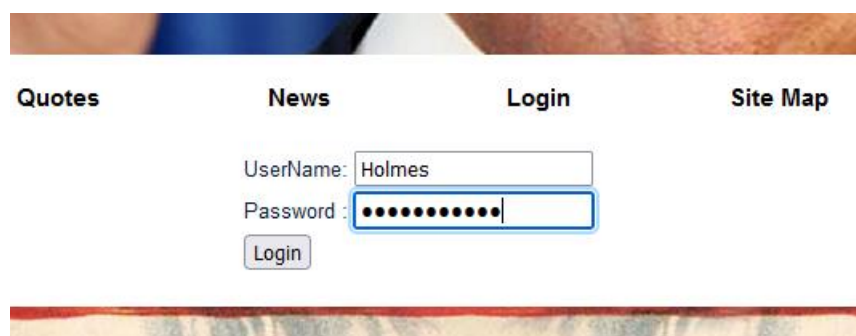
Fixing Plain Text Password

As we can see below, the password field type is text. By changing this to password type, it will hide the input.

```
<form action="ValidateLogin" method="post">
  <table>
    <tr><td>UserName: </td><td><input type="text" name="username" value="<%=username%>" /></td></tr>
    <tr><td>Password :</td><td><input type="text" name="password" value="<%=password%>" /></td></tr>
    <tr><td><input type="submit" name="Login" value="Login"/></td></tr>
  </table>

  -----

  <table>
    <tr><td>UserName: </td><td><input type="text" name="username" value="<%=username%>" /></td></tr>
    <tr><td>Password :</td><td><input type="password" name="password" value="<%=password%>" /></td></tr>
    <tr><td><input type="submit" name="Login" value="Login"/></td></tr>
  </table>
```



Also, for better security, we can request that the input is of a certain format, but to do this we need to make sure that at the registration page we will ask the user of the same request.

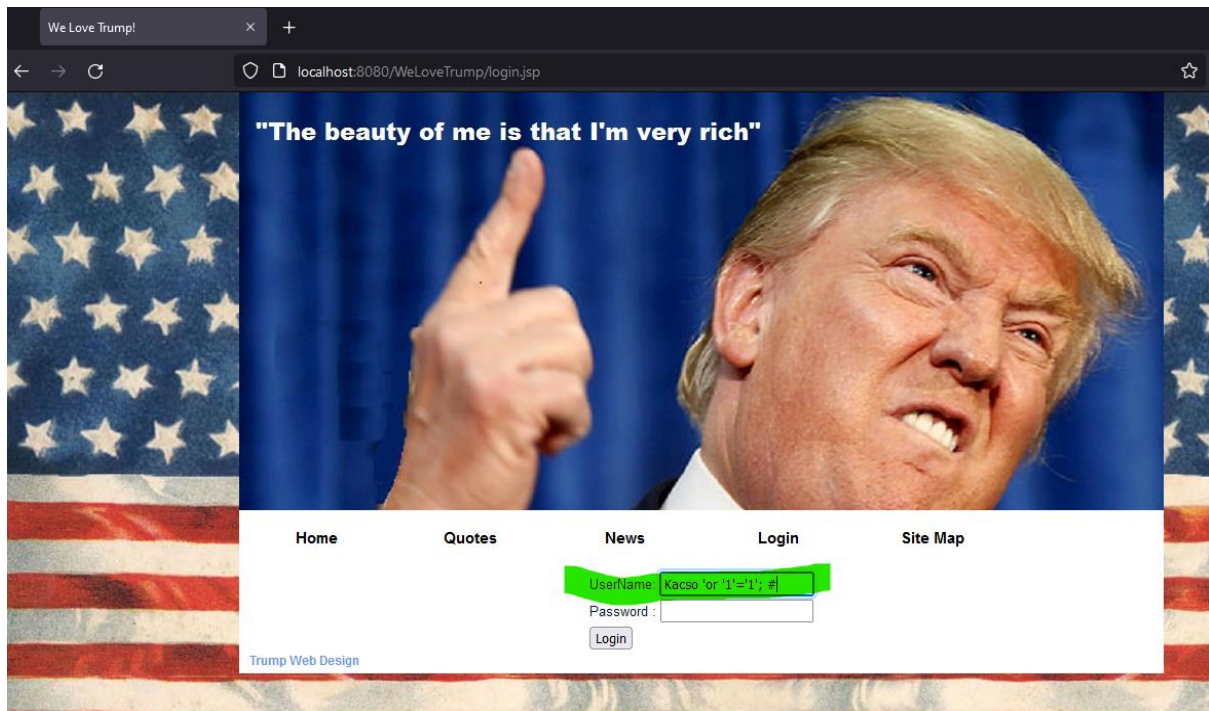
Regex can help against cross-site scripting too, as if you place a regex on the search field to make sure no special characters are accepted to be inputted, it can prevent someone from inputting a script into the search field.

Example of a regex inside the html:

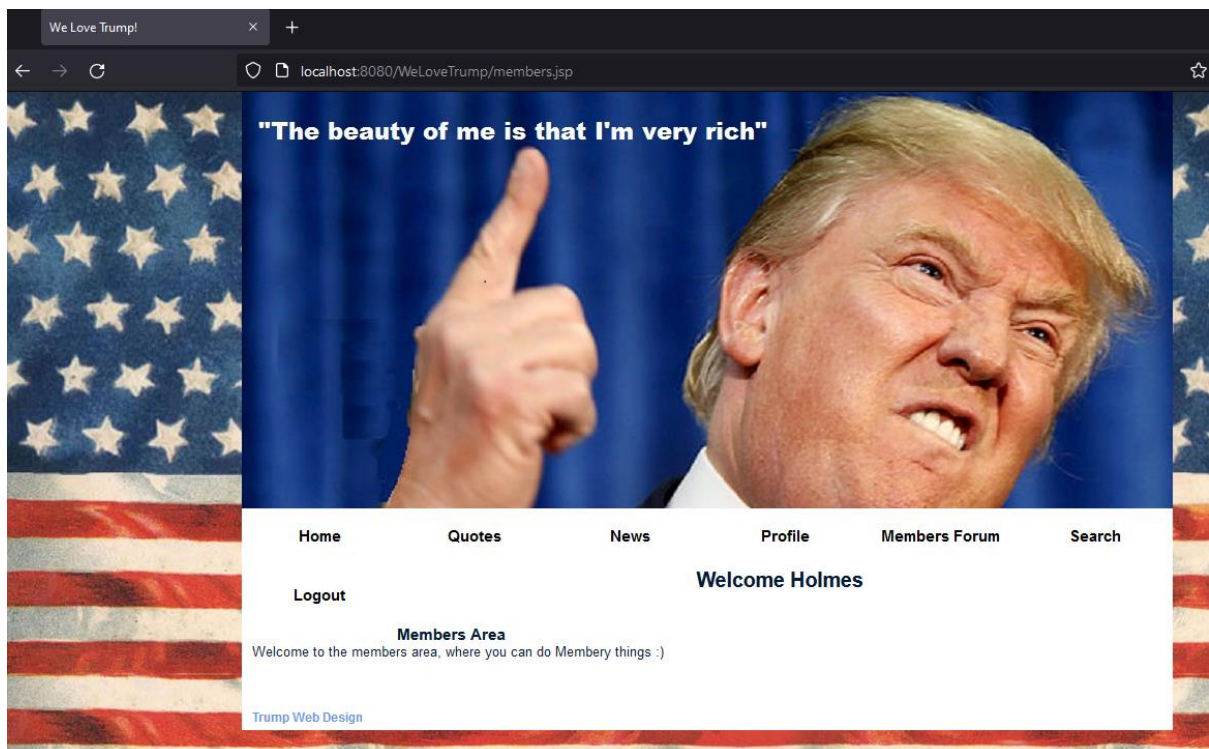
pattern="(?!.*\d)(?!.*[a-z])(?!.*[A-Z]).{8,}" title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters" required.

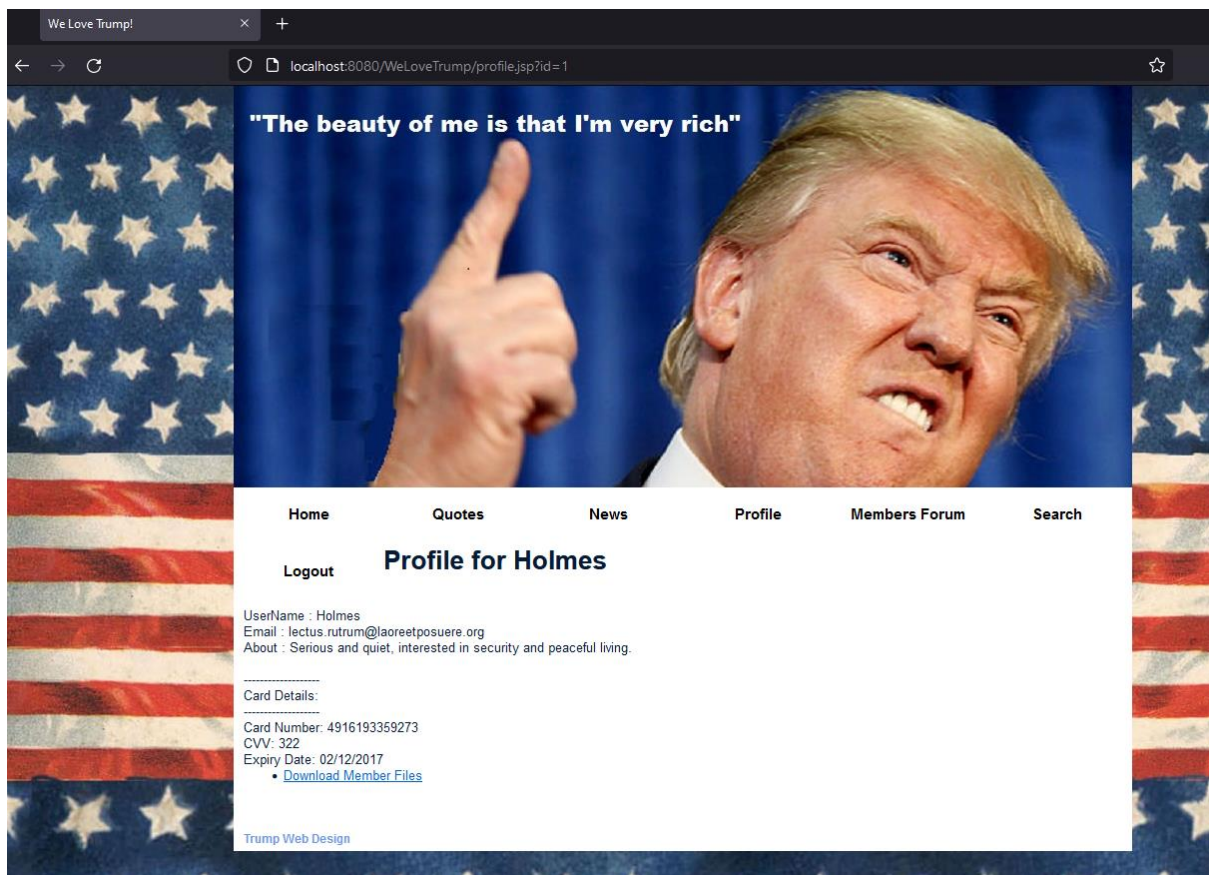
SQL injections

Next, I went to the login page and checked for SQL injection vulnerabilities. I used a simple method where I request a database query directly inputted in the Username field. If the field is not sanitized, it will take that query and log me in. I used my surname as a username, but even without using any username, the query will work.



As can be seen below, I was logged in and when I click on the profile tab, it will show me a user with the username Holmes and all the details associated with this user.





When checking the database, we can see that Holmes is the first username in it as we requested in the query inputted by us to return the 1st user.

id	username	password	email	about
1	Holmes	MEC15DBF3XD	lectus.rutrum@laoreetposuere.org	Serious and quiet, interested in security and peacefu...
2	Louis	PVO84UPH4JN	Mauris@elit.com	Quiet and reserved, interested in how and why thing...
3	Colin	UBN90KIN1MZ	mi.fringilla@purus.net	Loyal to his peers and to his internal value systems, ...
4	Cameron	WPX87QUOOTE	odio.Aliquam.vulputate@placeratorcilacus.net	Well-developed sense of space and function. Rich in...
5	Chadwick	TXE68KVQ4FO	Aliquam.erat@etruncQuisque.co.uk	Quiet, kind, and conscientious. Can be depended on...
6	Fritz	NWK85AIT6PR	dolor@Nuncsollicitudin.net	Quiet, serious, sensitive and kind. Do not like confli...
7	Troy	BER80OFS8RR	molestie@scelerisque lorem.edu	Loyal and faithful. Extremely well-developed senses...
8	Scott	IPB44IHJ1VM	nunc.sed@orciin consequat.net	Quietly forceful, original, and sensitive. Tend to stick...
9	Gage	SRV32VJV3FS	mi.lacinia@per.co.uk	Extremely intuitive about people, and concerned for...
10	Wylie	RUP65MFR0GD	quam.elementum@magnis.ca	Quiet, reflective, and idealistic. Interested in serving...
11	Jakeem	FXL38OBR0AC	sollicitudin@variusNamporrtitor.com	lectus.
12	Aquila	VXD91COJ0VP	Etiam.ligula.tortor@aliquetdiam.com	Nam nulla magna,

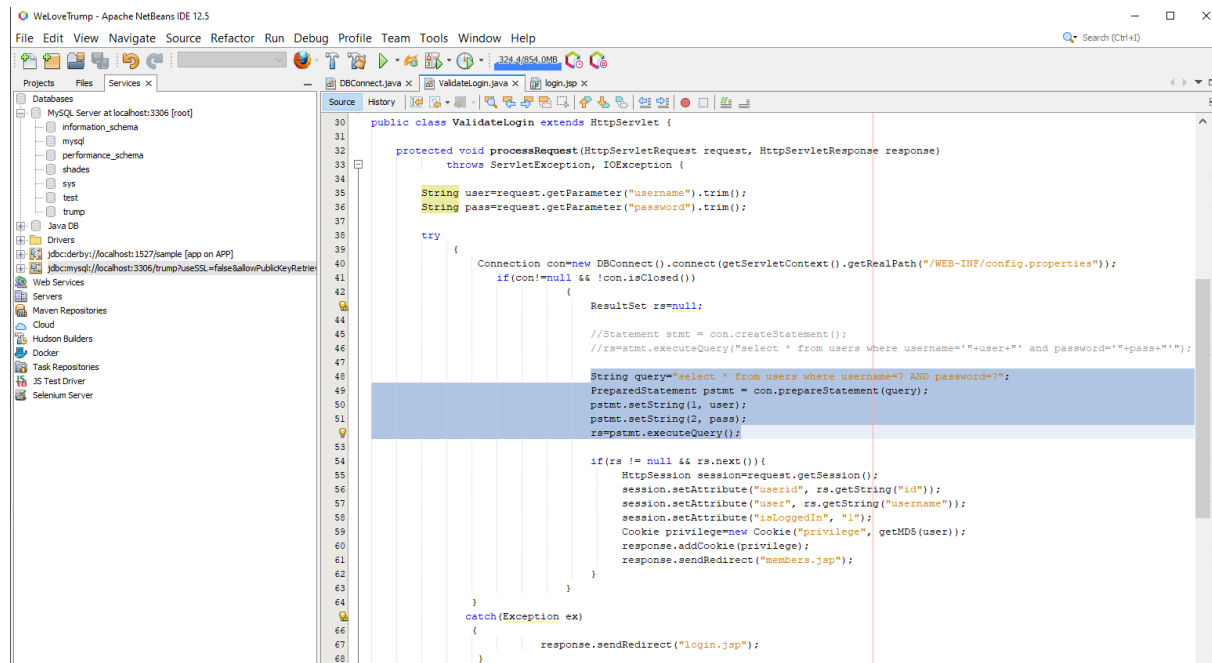
Also, the # at the end of that query that we inputted in the username field, is to ignore everything else after it. This way the original code below that is in place to request the username and password of a user is ignored.

```
Statement stmt = con.createStatement();
rs=stmt.executeQuery("select * from users where username='"+user+"' and password='"+pass+"'");
if(rs != null && rs.next()){
```

Using this method, other queries can be executed too. For example, dropping (deleting) an entire table from the database.

Fixing SQL injections

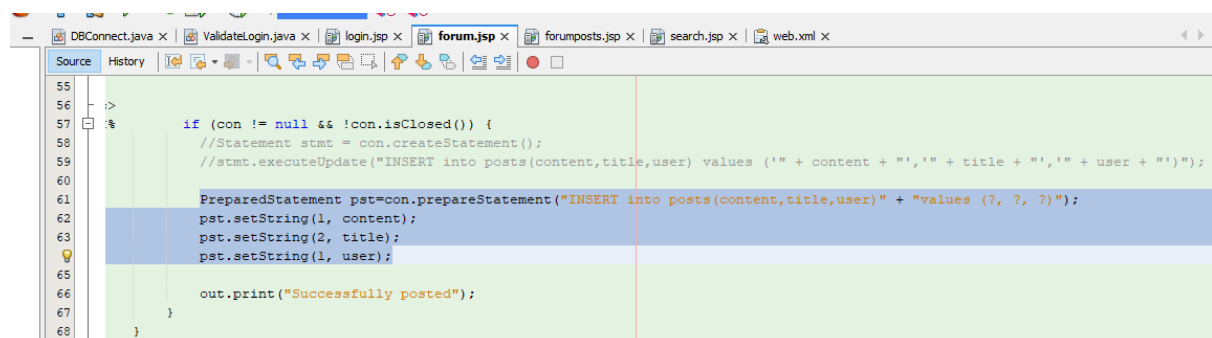
There are different ways to fix this, but the best way and most recommended way is to introduce a Prepared Statement. This works by separating the SQL queries from the data that is being called from the database. First, the database parses the given query and allocates space for the parameters. In our case is the '?' in the code below. After that, the data is being passed to these parameters.



```
30 public class ValidateLogin extends HttpServlet {
31
32     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33         throws ServletException, IOException {
34
35         String user=request.getParameter("username").trim();
36         String pass=request.getParameter("password").trim();
37
38         try
39         {
40             Connection con=new DBConnect().connect(getServletContext().getRealPath("/WEB-INF/config.properties"));
41             if(con!=null && !con.isClosed())
42             {
43                 ResultSet rs=null;
44
45                 //Statement stmt = con.createStatement();
46                 //rs=stmt.executeQuery("select * from users where username='"+user+"' and password='"+pass+"'");
47
48                 String query="select * from users where username=? AND password=?";
49                 PreparedStatement pstmt = con.prepareStatement(query);
50                 pstmt.setString(1, user);
51                 pstmt.setString(2, pass);
52                 rs=pstmt.executeQuery();
53
54                 if(rs != null && rs.next()){
55                     HttpSession session=request.getSession();
56                     session.setAttribute("userid", rs.getString("id"));
57                     session.setAttribute("user", rs.getString("username"));
58                     session.setAttribute("isLoggedIn", "1");
59                     Cookie privilege=new Cookie("privilege", getMDS(user));
60                     response.addCookie(privilege);
61                     response.sendRedirect("members.jsp");
62                 }
63             }
64         }
65         catch(Exception ex)
66         {
67             response.sendRedirect("login.jsp");
68         }
69     }
```

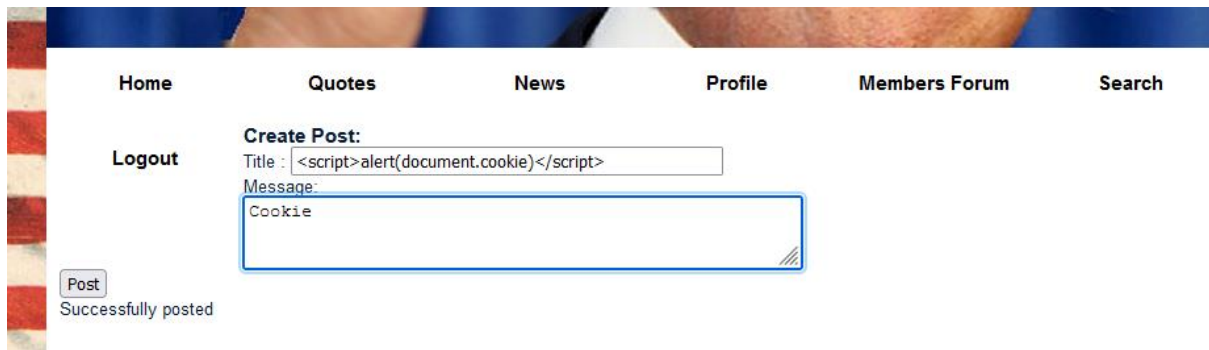
Cross-site scripting (XSS)

Same as above, we can secure the forum's post form as well with a prepared statement.



```
55
56
57     if (con != null && !con.isClosed()) {
58         //Statement stmt = con.createStatement();
59         //stmt.executeUpdate("INSERT into posts(content,title,user) values ('" + content + "','" + title + "','" + user + "')");
60
61         PreparedStatement pstmt=con.prepareStatement("INSERT into posts(content,title,user)" + "values (?, ?, ?)");
62         pstmt.setString(1, content);
63         pstmt.setString(2, title);
64         pstmt.setString(3, user);
65
66         out.print("Successfully posted");
67     }
68 }
```

After injecting a script in the title of the post, it is not run, as visible when we inspect the source code on the page.



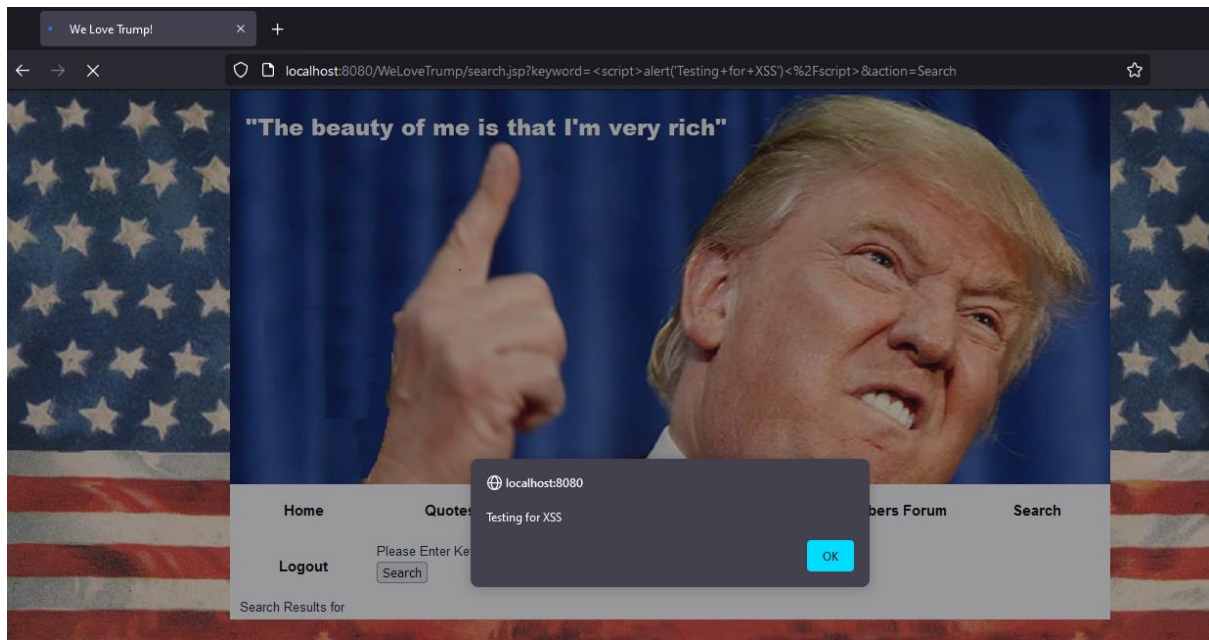
```

1
2
3
4
5
6 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7 <html xmlns="http://www.w3.org/1999/xhtml">
8   <head>
9     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
10    <link rel="stylesheet" type="text/css" href="style.css" />
11    <title>We Love Trump!</title>
12  </head>
13  Hello Holmes, Welcome to Our Forum !
14  <body>
15    <div id="container">
16      <div id="mainpic">
17      </div>
18
19      <div id="menu">
20        <ul>
21          <li class="menuitem"><a href="index.jsp">Home</a></li>
22          <li class="menuitem"><a href="quotes.jsp">Quotes</a></li>
23          <li class="menuitem"><a href="news.jsp">News</a></li>
24          <li class="menuitem"><a href="profile.jsp?id=1">Profile</a></li>
25          <li class="menuitem"><a href="forum.jsp">Members Forum</a></li>
26          <li class="menuitem"><a href="search.jsp">Search</a></li>
27          <li class="menuitem"><a href="ValidateLogout">Logout</a></li>
28        </ul>
29      </div>
30
31      <div id="content">
32        <h3>Create Post:</h3>
33        <form action="forum.jsp" method="POST">
34          Title : <input type="text" name="title" value="" size="50"/><br/>
35          Message: <br/><textarea name="content" rows="2" cols="50"></textarea>
36          <input type="hidden" name="user" value="Holmes"/><br/>
37          <input type="submit" value="Post" name="post"/>
38        </form>
39
40        Successfully posted
41        <p>&nbsp;</p>
42        <p>&nbsp;</p>
43        <p>&nbsp;</p>
44        <h3>List of Posts:</h3>
45        <table border="1" width="80%">
46          <tr><td><a href="forumposts.jsp?postId=1">The right one?</a></td><td> - Posted By Nigel</td></tr>
47          <tr><td><a href="forumposts.jsp?postId=2">Hairy moment</a></td><td> - Posted By Keegan</td></tr>
48          <tr><td><a href="forumposts.jsp?postId=3">Everything in moderation</a></td><td> - Posted By Steven</td></tr>
49        </table>
50
51
52        <div id="footer"><h3><a href="http://www.trump.com/">Trump Web Design</a></h3></div>
53      </div>
54    </div>
55  </body>
56

```

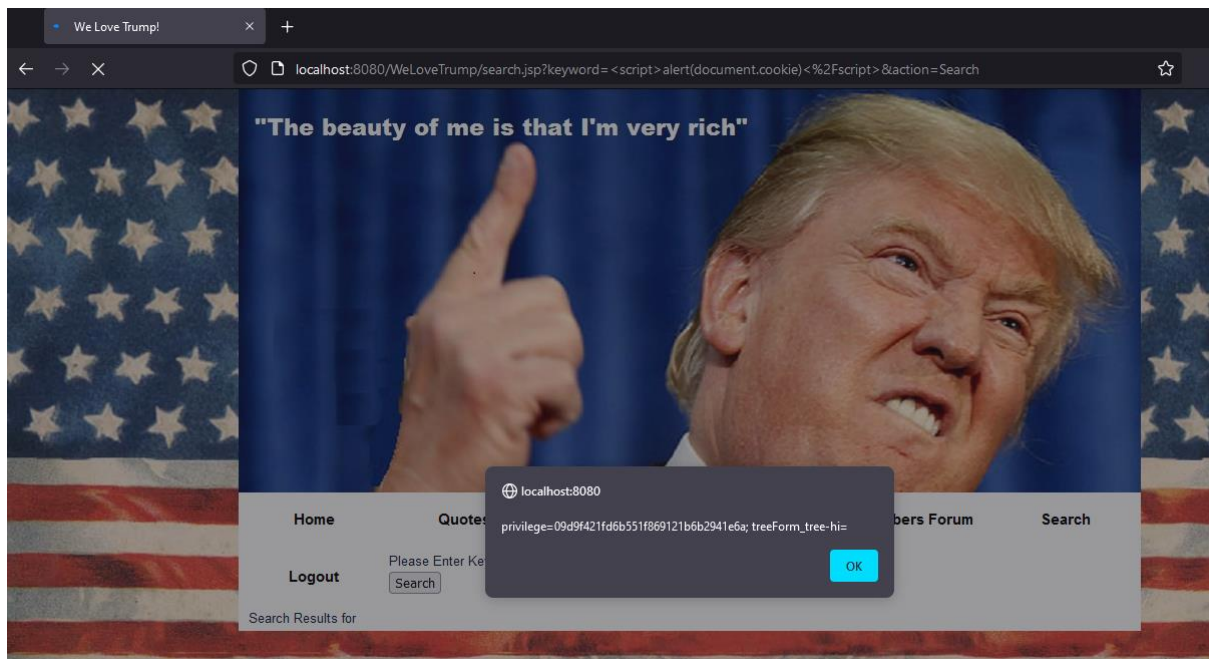
Another way of testing for cross-site scripting vulnerabilities is by introducing a script into the search field.

A script like `<script>alert('Testing for XSS')</script>` would display a pop-up message if the search field is vulnerable.



As we can see, the search field is vulnerable.

The attacker could use this to steal the session ID of the user by running a script like **`<script>alert(document.cookie)</script>`**



Inspecting the source code of this page, we can see that the script has been run.

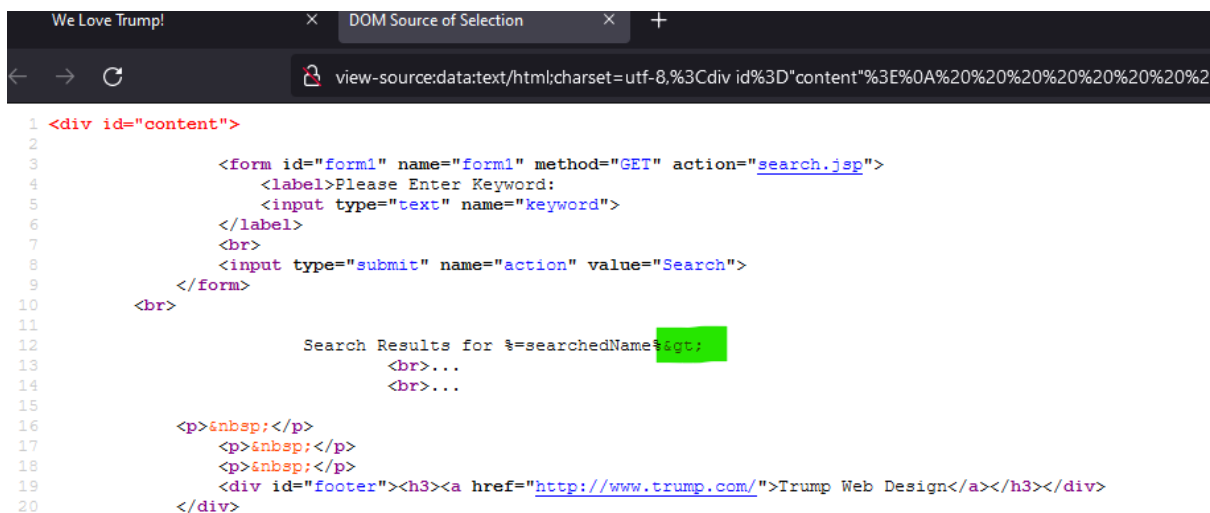
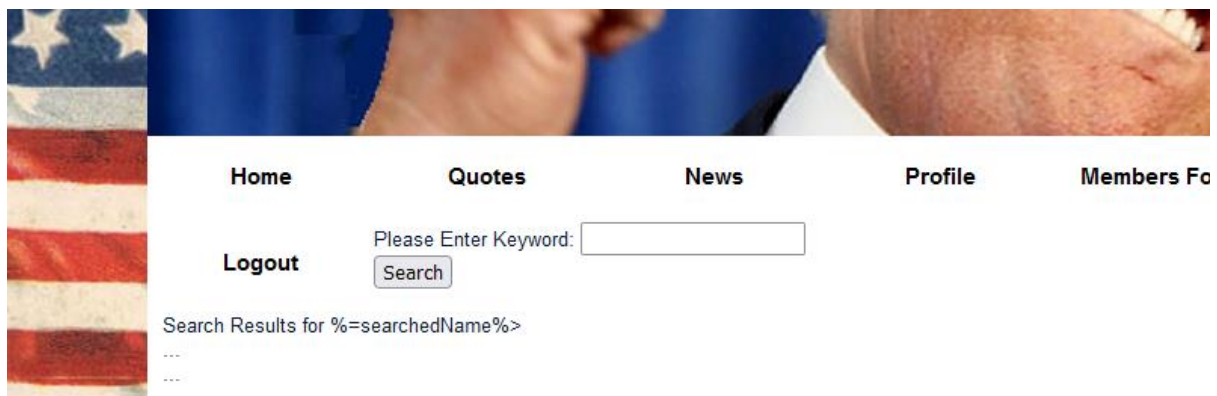
```

26      <li class="menuitem"><a href="ValidateLogout">Logout</a></li>
27    </ul>
28  </div>
29 </div>
30
31 <div id="content">
32
33   <form id="form1" name="form1" method="GET" action="search.jsp">
34     <label>Please Enter Keyword:
35     <input type="text" name="keyword"/>
36   </label>
37   <br>
38   <input type="submit" name="action" value="Search"/>
39 </form>
40 <br/>
41
42   Search Results for <script>alert('Testing for XSS')</script>
43   <br/>...
44   <br/>...
45
46   <p>&nbsp;</p>
47   <p>&nbsp;</p>
48   <p>&nbsp;</p>
49   <div id="footer"><h3><a href="http://www.trump.com/">Trump Web Design</a></h3></div>
50 </div>
51 </div>
52
53

```

Fixing Cross-Site Scripting Issues

We can load and use a library called JSTL, which contains the TagLib library, which essentially replaces HTML characters (in this case ">"), with HTML character entities (in this case ">"), which causes the script to break and won't run.



Another way is by downloading a jar file called ESAPI from the OWASP website, which contains an HTML encoder, making sure, like the TagLib that the HTML characters are being encoded with special characters. I have used this on the forum posting page, in addition to the prepared statement presented above.

```
<%
    if (request.getParameter("post") != null) {
        String user = ESAPI.encoder().encodeForHTML(request.getParameter("user"));
        String content = ESAPI.encoder().encodeForHTML(request.getParameter("content"));
        String title = ESAPI.encoder().encodeForHTML(request.getParameter("title"));

    }

    if (con != null && !con.isClosed()) {
        //Statement stmt = con.createStatement();
        //stmt.executeUpdate("INSERT into posts(content,title,user) values ('" + content + "','" + title + "','"

        PreparedStatement pst=con.prepareStatement("INSERT into posts(content,title,user)" + "values (?, ?, ?)")
        pst.setString(1, content);
        pst.setString(2, title);
        pst.setString(1, user);

        out.print("Successfully posted");
    }
}
```

As we can see the post is being created, but the script itself is not run. While I used the TagLib above on the search field, I used the ESAPI jar file in the posts to showcase both ways of fixing cross-site vulnerabilities.

List of Posts:

The right one?	- Posted By Nigel
Hairy moment	- Posted By Keegan
Everything in moderation	- Posted By Steven
Post after ESAPI	- Posted By Holmes
<script>alert('Testing after ESAPI')</script>	- Posted By Holmes

Trump Web Design

And if we inspect the source code of the script, we will notice that the HTML tags are being replaced with HTML entities, to break the script. Even though it is the 5th post, it has a postid=9 because I had to delete the script entries from the database, to stop them from running anytime I was opening the forums post page.

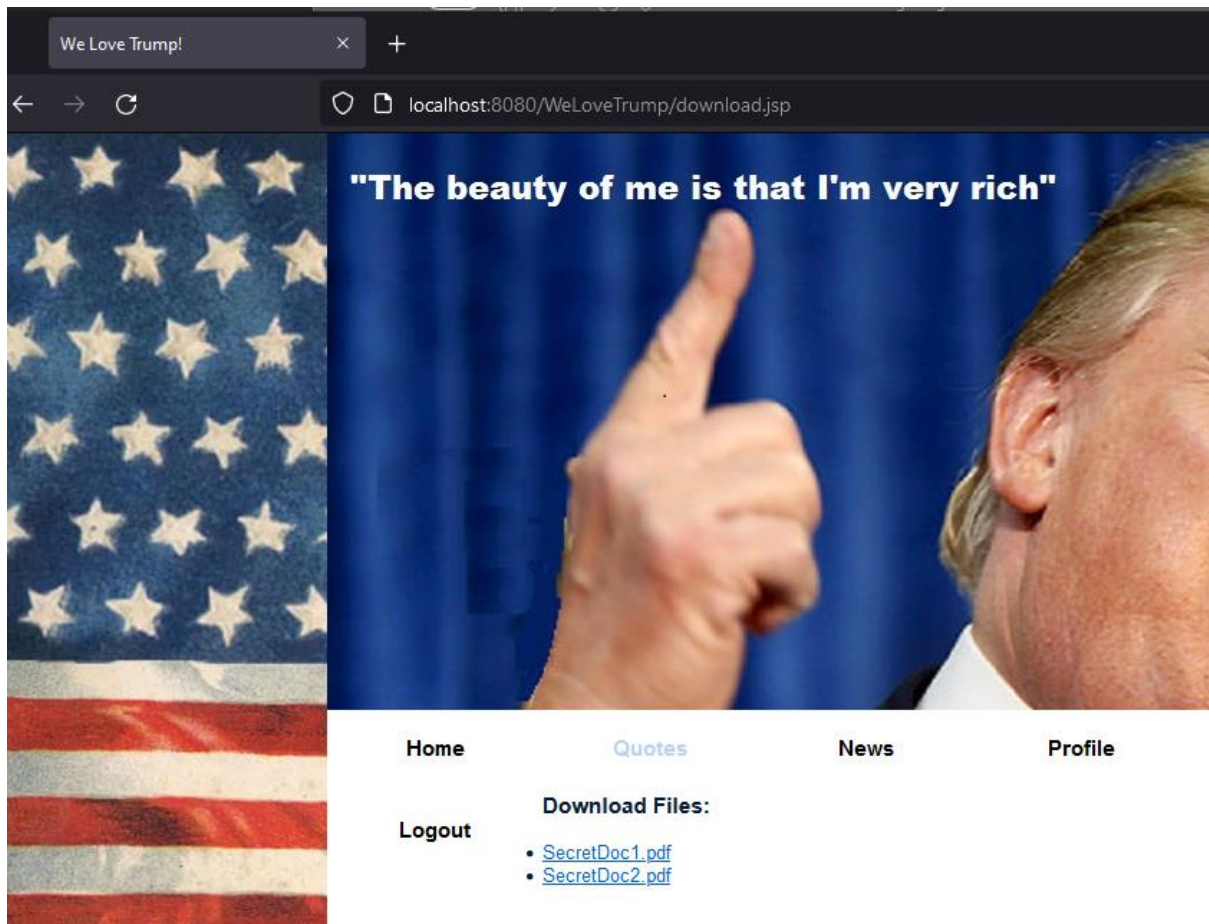
We Love Trump! x DOM Source of Selection x +

view-source:data:text/html;charset=utf-8,%3Ca href%3D"forumposts.jsp%3Fpostid%3D9"%3E%EF

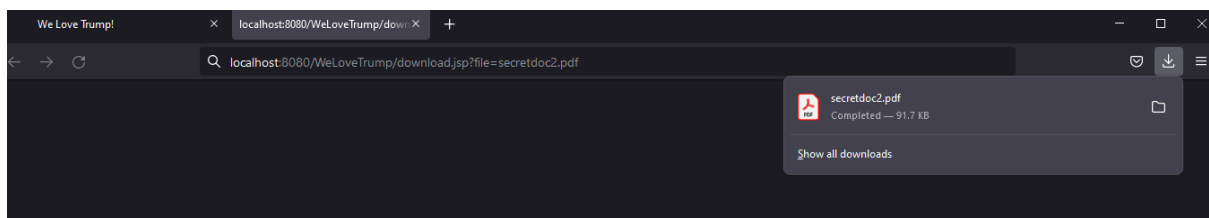
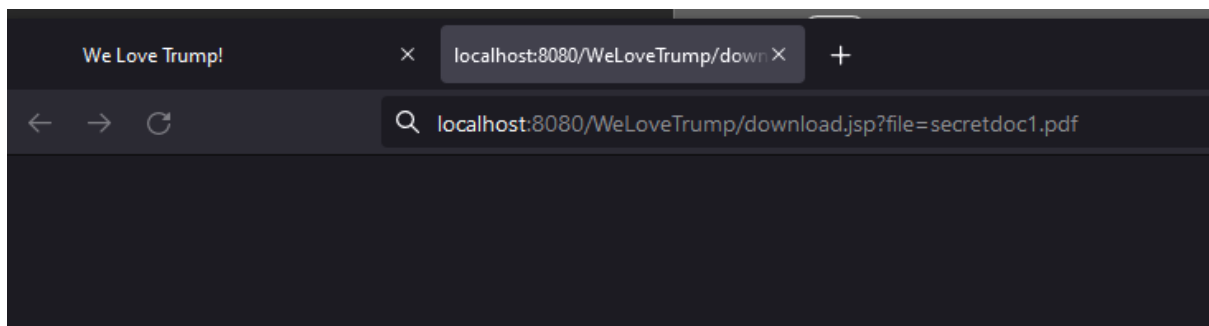
```
1 <a href="forumposts.jsp?postid=9">&lt;script&gt;alert("Testing after ESAPI")&lt;/script&gt;</a>
```

IDOR (Insecure Direct Object References)

When we access the profile page, we can see that there is a download link called "Download Members Files". If we open that, we can see 2 documents in it, called SecretDoc1.pdf and SecretDoc2.pdf. By right-clicking on the first one, and choosing to open it in a different tab, we can modify its link and we can gain access to other files.



In this example, we opened the link for the first doc and modified it to download the second document. If we would be aware of other file names stored on this site, we could download them all without having direct access to them.



Fixing IDOR

To fix this issue, we will use the ESAPI library again, by creating a new java class called DownloadFiles.java which will be used to validate the files in the download.jsp file. This is a method for AccessReferenceMap of the ESAPI library.

```
5 package validate;
6
7 import java.util.HashSet;
8 import java.util.Set;
9 import org.owasp.esapi.AccessReferenceMap;
10 import org.owasp.esapi.reference.RandomAccessReferenceMap;
11
12 /**
13  *
14  * @author digig
15  */
16 public class DownloadFiles {
17     public static AccessReferenceMap getDownloadFiles()
18     {
19         Set filesSet=new HashSet();
20         filesSet.add("secretdoc1.pdf");
21         filesSet.add("secretdoc2.pdf");
22         return new RandomAccessReferenceMap(filesSet);
23     }
24 }
```

Once we created the new java class, we call it in, in the download.jsp file by assigning it to a variable called map. If this variable is called, it will download the file, or if it's null it won't. This is a direct reference to the download file.

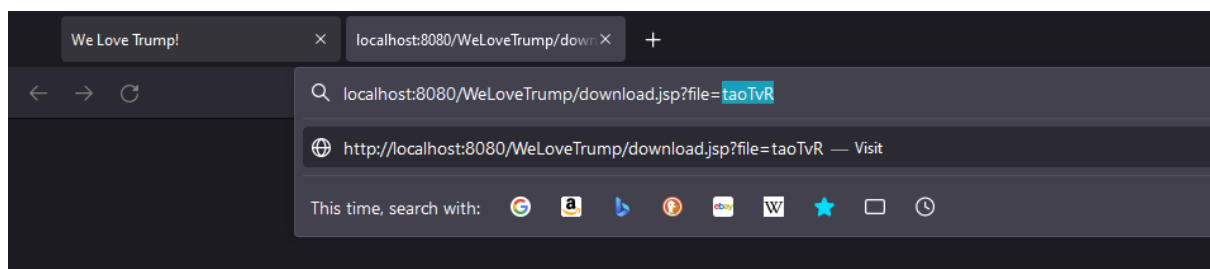
```
33 <div id="content">
34 <%
35     AccessReferenceMap map=null;
36     if(session.getAttribute("dwnldFile")==null)
37     {
38         map=DownloadFiles.getDownloadFiles();
39         session.setAttribute("dwnldFile", map);
40     }
41     else
42     {
43         map=(AccessReferenceMap) session.getAttribute("dwnldFile");
44     }
45 }
```

Now to make it an indirect reference, so the file name in the URL will appear encoded, we will wrap the files inside it. Also, a direct reference to the filePath is being called.

```
<h3> Download Files: </h3><br/>
<ul>
<li><a href="download.jsp?file=<%=map.getIndirectReference("secretdoc1.pdf")%>"> SecretDoc1.pdf </a> </li>
<li><a href="download.jsp?file=<%=map.getIndirectReference("secretdoc2.pdf")%>"> SecretDoc2.pdf </a></li>
</ul>
<p>&nbsp;</p>
```

```
{
    filePath=(String) map.getDirectReference(filePath);
}
```

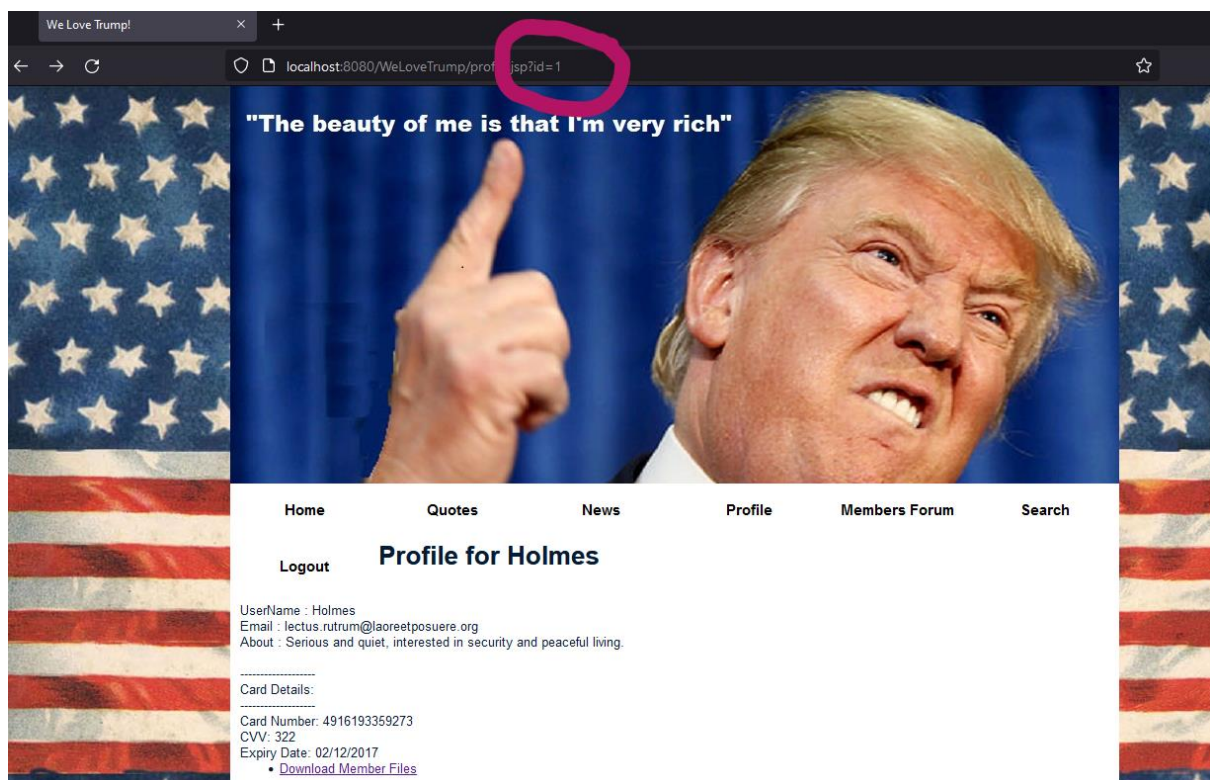
Once done, the name of the files will appear encoded in the URL.

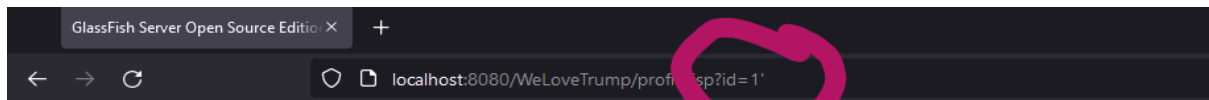


Security Misconfiguration

Just by modifying the URL slightly, we can get errors that could reveal important information. Sometimes we would be able to see database queries within those errors which can be exploited.

Below are screenshots of these errors on 2 different pages. The profile page shows the id of the user, and the next one is from the forum page showing the post id. By introducing a single quote mark at the end of it, an error page is given.





HTTP Status 500 - Internal Server Error

type Exception report

message Internal Server Error

description The server encountered an internal error that prevented it from fulfilling this request.

exception

org.apache.jasper.JasperException: PWC6033: Error in Javac compilation for JSP

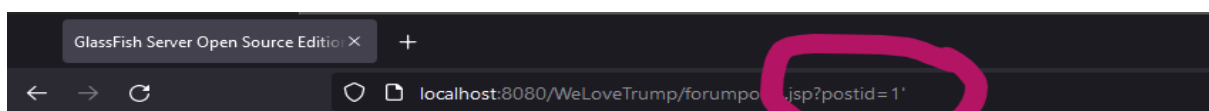
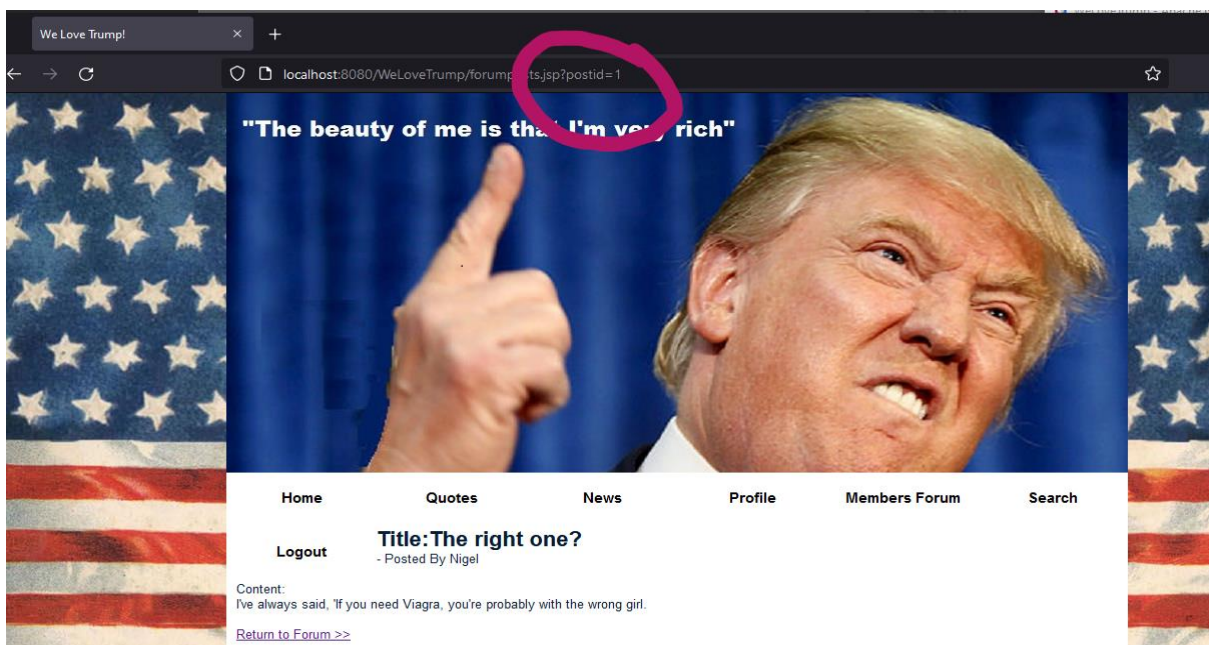
PWC6199: Generated servlet error:
source value 1.5 is obsolete and will be removed in a future release

PWC6199: Generated servlet error:
target value 1.5 is obsolete and will be removed in a future release

PWC6199: Generated servlet error:
To suppress warnings about obsolete options, use -Xlint:-options.

PWC6199: Generated servlet error:
'try' without 'catch' or 'finally'

PWC6197: An error occurred at line: 39 in the jsp file: /profile.jsp



HTTP Status 500 - Internal Server Error

type Exception report

message Internal Server Error

description The server encountered an internal error that prevented it from fulfilling this request.

exception

javax.servlet.ServletException: java.sql.SQLException: You have an error in your SQL syntax; c

root cause

java.sql.SQLException: You have an error in your SQL syntax; check the manual that corresponds

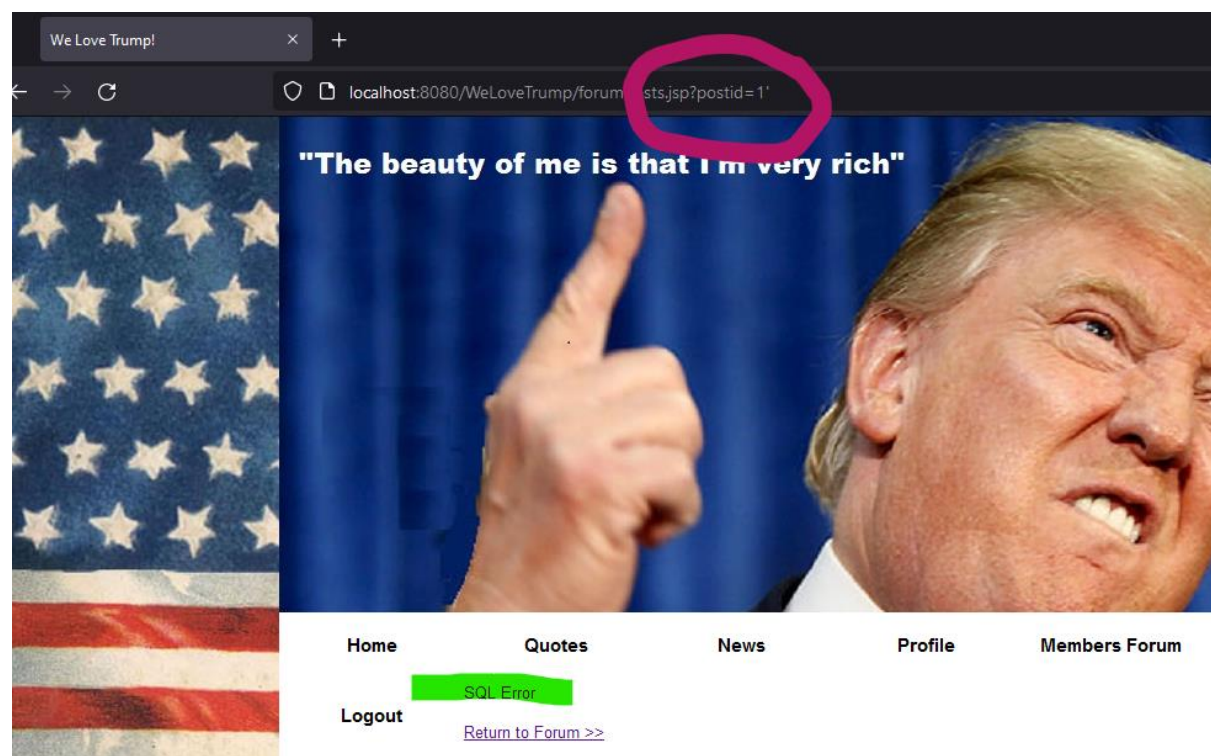
note The full stack traces of the exception and its root causes are available in the GlassFish Server Open Source Edition 5.0 logs.

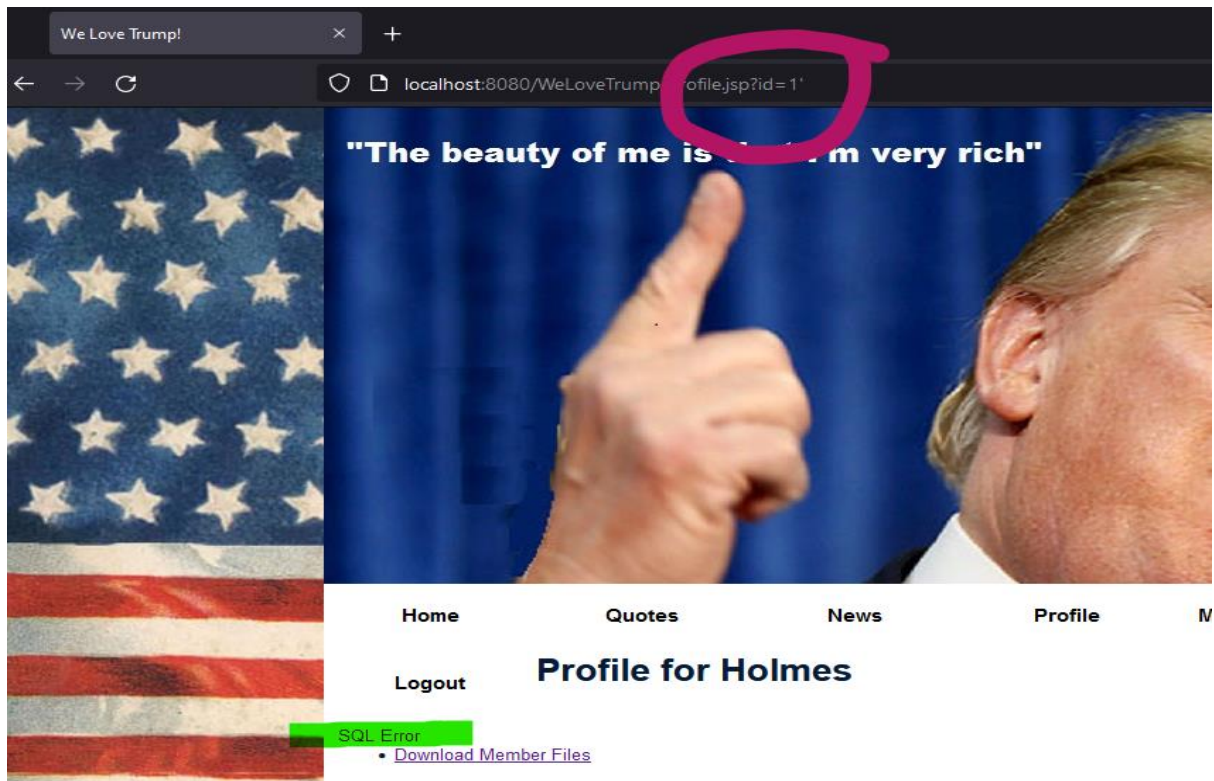
GlassFish Server Open Source Edition 5.0

Fixing Security Misconfigurations

To fix this error, we can introduce a try-catch, which will catch these errors and display them for us, instead of revealing the error page, thus not revealing information to the attacker.

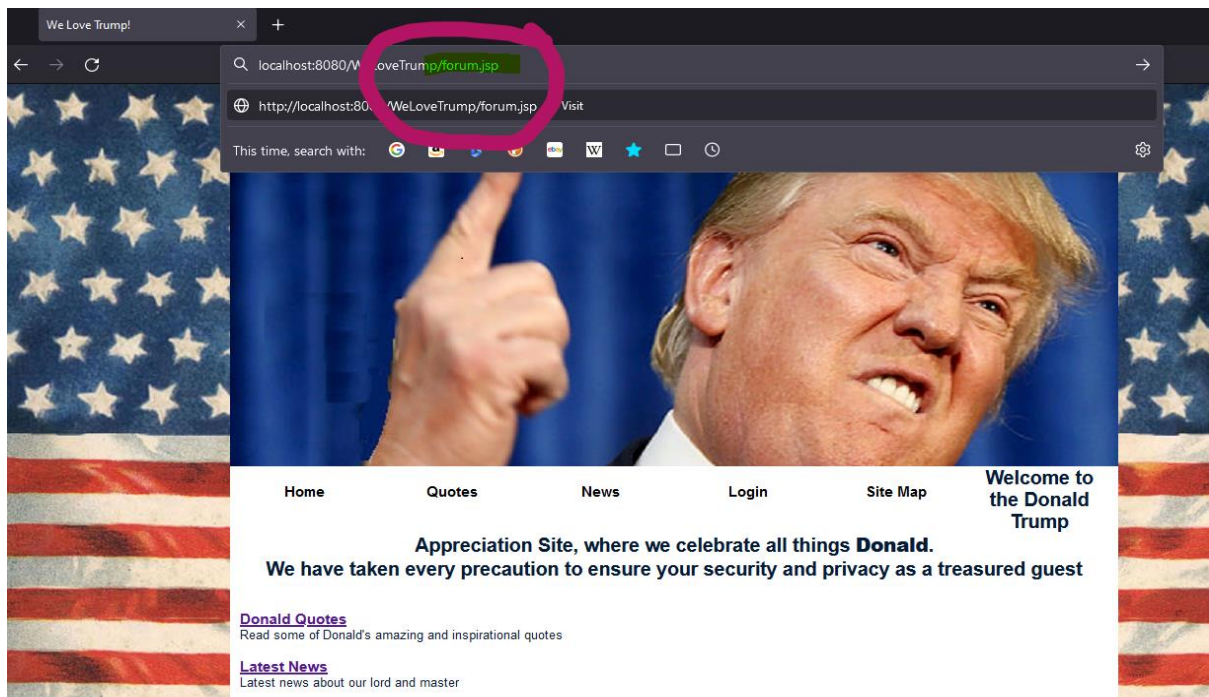
```
Connection con = new DBConnect().connect(getServletContext().getRealPath("/WEB-INF/config.properties"));
try
{
    String postid = request.getParameter("postid");
    if (postid != null) {
        Statement stmt = con.createStatement();
        ResultSet rs = null;
        rs = stmt.executeQuery("select * from posts where id=" + postid);
        if (rs != null && rs.next()) {
            out.print("<b style='font-size:22px'>Title:" + rs.getString("title") + "</b>");
            out.print("<br/>- Posted By " + rs.getString("user"));
            out.print("<br/><br/>Content:<br/>" + rs.getString("content"));
        }
    } else {
        out.print("ID Parameter is Missing");
    }
}
catch(SQLException e)
{
    out.print("SQL Error");
}
catch(Exception e)
{
    out.print("Error message");
}
```





Unvalidated URL's

Next, we will check for unvalidated URL's. That means that if we are on the homepage, and we modify the URL to bring us to the forum page, which is a page that should be accessed only after you sign in, it shouldn't let us navigate to it. It should either redirect us to the login page or at least give an error message saying that is not possible to view that page.



Unfortunately, we are redirected to the forum.jsp page after just modifying the URL.

Fixing Unvalidated URL's

To fix this vulnerability, we will have to make sure that any page that should be accessed only after we logged in, can't be accessed otherwise. In our case is the profile page, forum page, forum posts page, search page, member's page and download page.

To do this, we will add a code to check if the user is logged in, against the variable isLoggedIn from the ValidateLogin.java file, and if they are then let them navigate to that page, if not, redirect them to the login page.

```
</head>
<%
String validateUser=null;
if(session.getAttribute("isLoggedIn") !=null && session.getAttribute("isLoggedIn").equals("1"))
{
    out.print(validateUser);
    validateUser = session.getAttribute("user").toString();
}
else{response.sendRedirect("login.jsp");}
%>

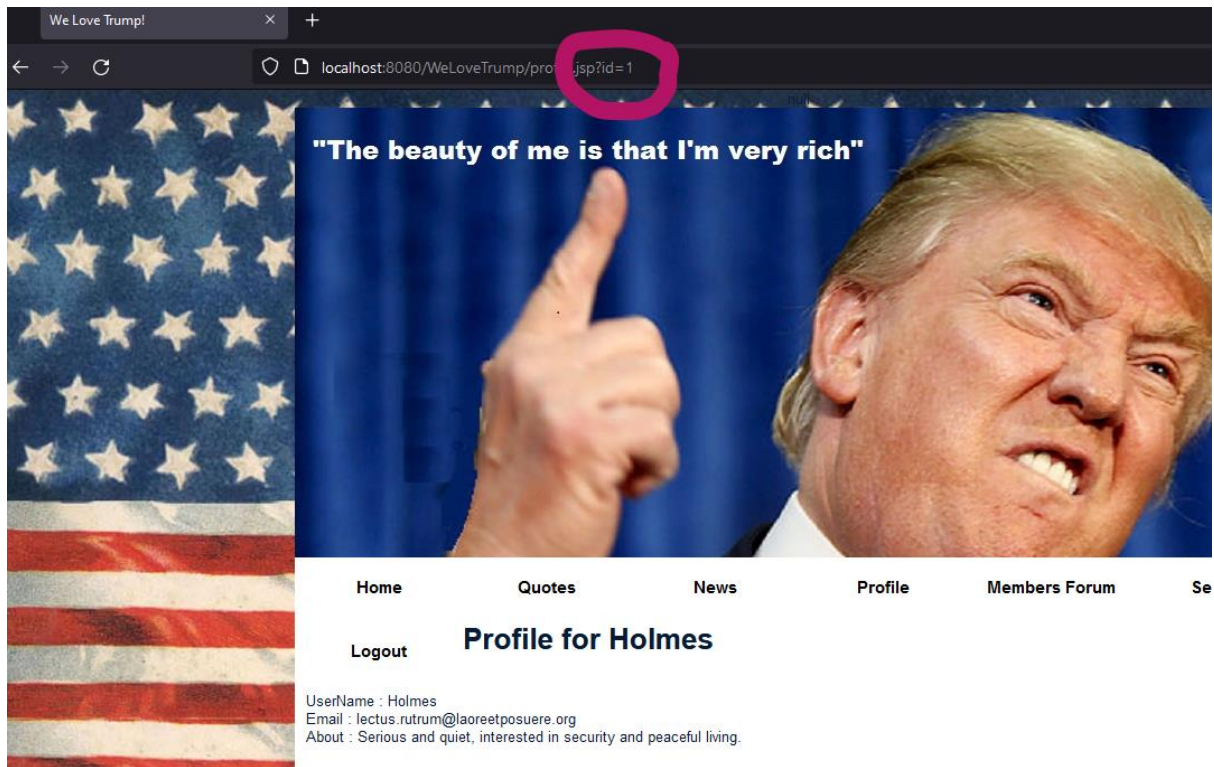
<body>
<div id="container">
```

Then at the top of each page, we can validate for the user before giving them access.

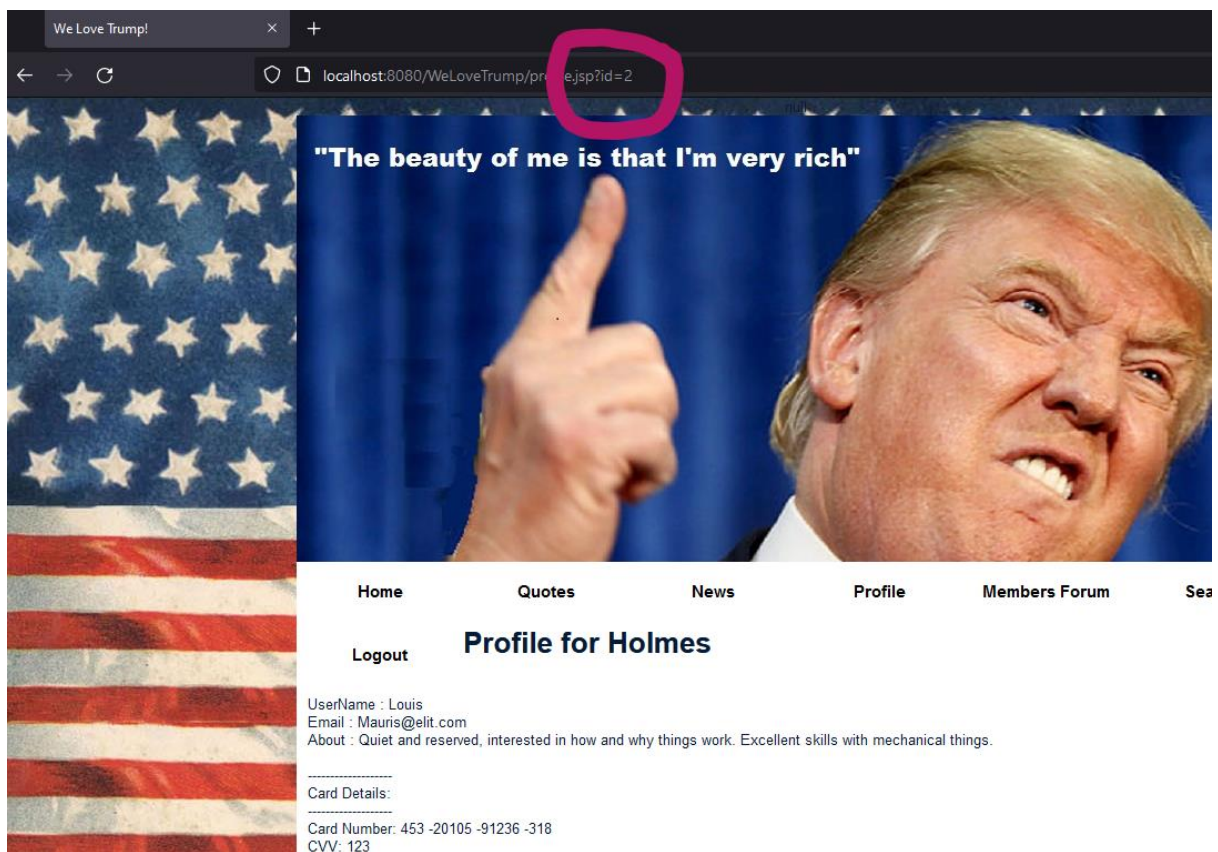
```
<div id="content">
    <h1>Profile for <%=validateUser%></h1>
    <p>&nbsp;</p>
    <p>&nbsp;</p>
```

URL Modification

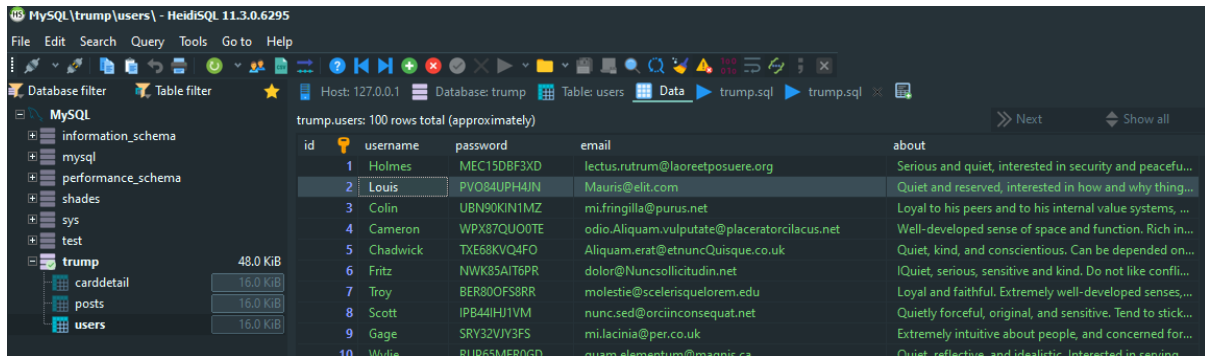
We log in to the first user from the database called Holmes, and we can see that the user id=1 in the URL.



By changing that to another ID, we can see that we are redirected to another user that has that ID in the database.



Let's check the database to see if Louis is the user associated with id=2.

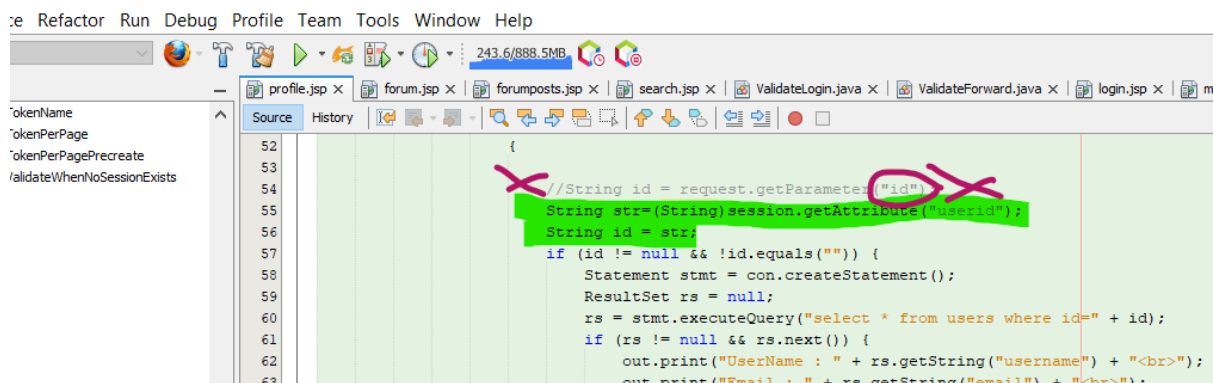


id	username	password	email	about
1	Holmes	MEC15DBF3XD	lectus.rutrum@laoreetposuere.org	Serious and quiet, interested in security and peacefu...
2	Louis	PVO84UPH4JIN	Mauris@elit.com	Quiet and reserved, interested in how and why thing...
3	Colin	UBN90KIN1MZ	mi.fringilla@purus.net	Loyal to his peers and to his internal value systems, ...
4	Cameron	WPX87QUO0TE	odio.Aliquam.vulputate@placeratorcilacus.net	Well-developed sense of space and function. Rich in...
5	Chadwick	TXE68KVQ4FO	Aliquam.erat@etruncQuisque.co.uk	Quiet, kind, and conscientious. Can be depended on...
6	Fritz	NWK85AIT6PR	dolor@Nuncsollicitudin.net	Quiet, serious, sensitive and kind. Do not like confli...
7	Troy	BER800FS8RR	molestie@scelerisque lorem.edu	Loyal and faithful. Extremely well-developed senses...
8	Scott	IPB44IH1VM	nunc.sed@orciinconsequat.net	Quietly forceful, original, and sensitive. Tend to stick...
9	Gage	SRV32VJY3FS	mi.lacinia@per.co.uk	Extremely intuitive about people, and concerned for...
10	Widie	BUH55MEBNDG	quam.elementum@maecenas	Quiet, reflective, and idealistic. Interested in sepi...

It is indeed. This is not acceptable as any user that is already registered with this website has access to other users' information.

Fixing the URL Modification vulnerability

To fix this all we need to do is change the requested parameter on the profile.jsp page, from "id" to "userid". The user ID parameter was already given an attribute in the ValidateLogin.java page. We just call in that attribute instead of the ID attribute. Also, we define the String id that was used in the previous code, to the new string "str" that was used to get the attribute of "userid".

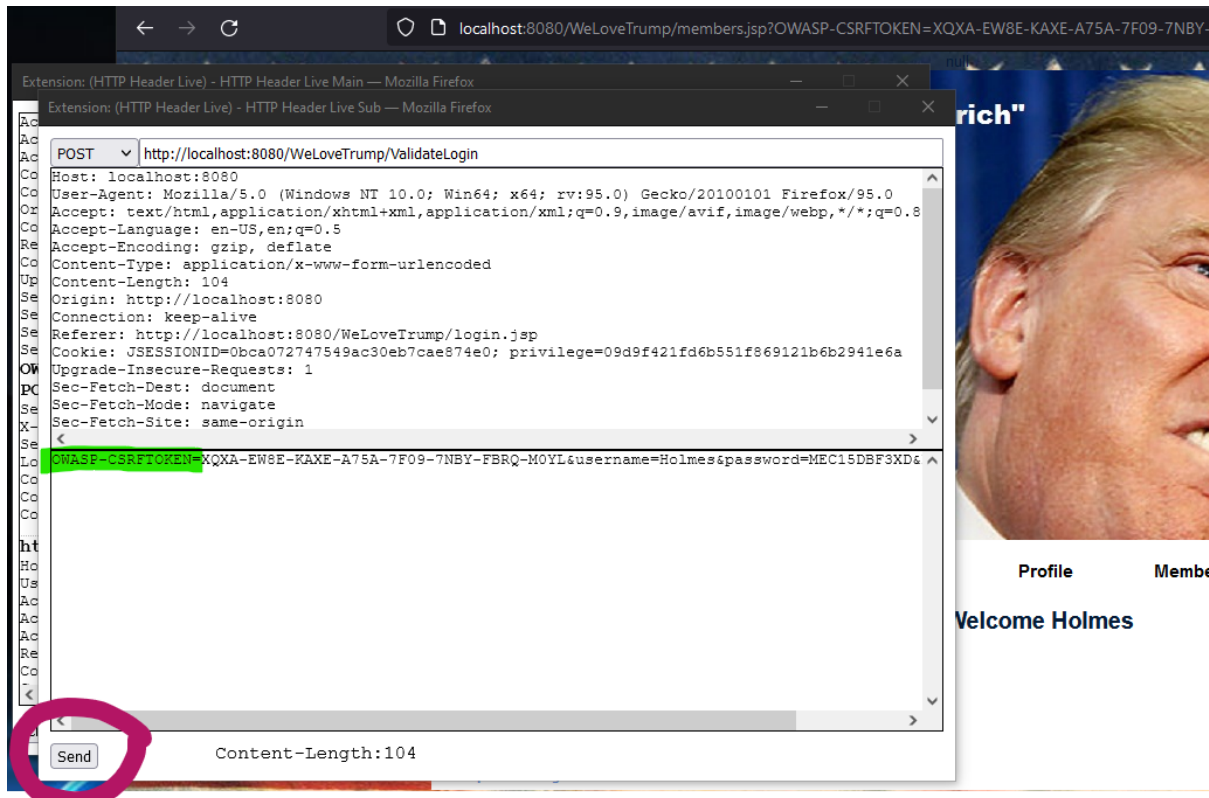


CSRF (Cross-Site Request Forgery)

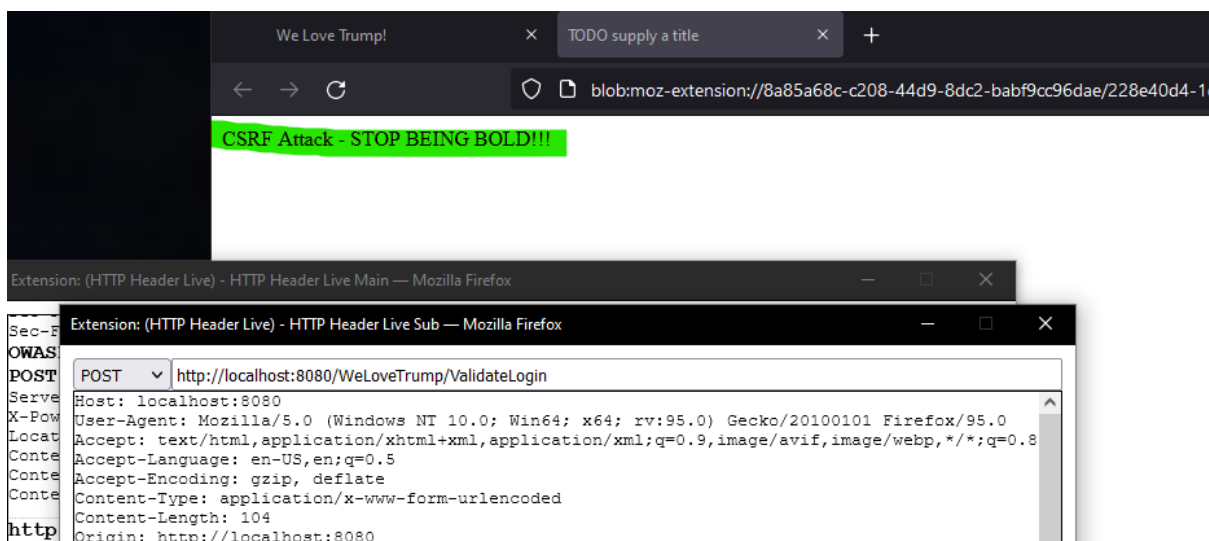
If we don't validate against CSRF attacks, we can have an attacker take our session ID from the cookies and forward them to their server and reuse the session token to log in to the victim's profile. This type of attack is especially good if placed on the login page.

Fixing CSRF

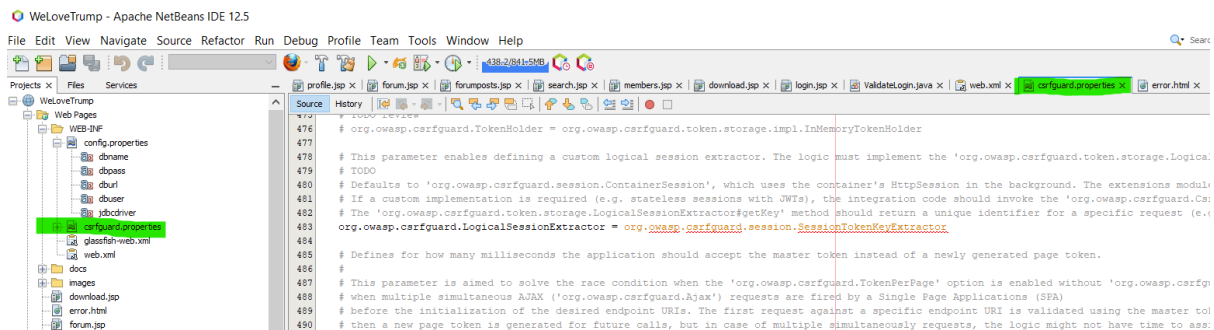
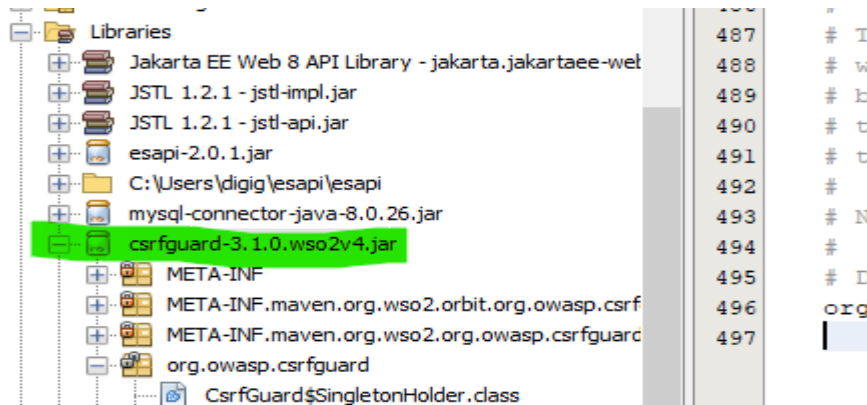
As seen below, a CSRF validation was implemented, using a CSRF token, and when we try to resend or forward the request for a second time, it will be stopped, and an error page is shown.



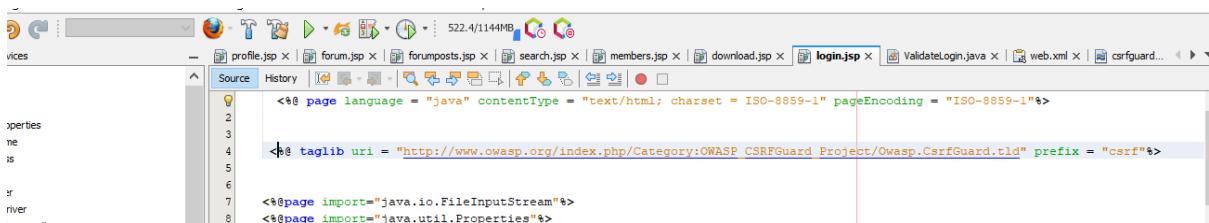
In our case we redirected the error to another HTML page that we created for this specific error, this way making sure that no vulnerable information is being exposed in the error page, but instead a simple blank page with just an error message.



To do this, the csrfguard-3.1.0.wso2v4.jar file was added to the library and the csrfguard.properties file was created to import all the properties into it.



After that, we added a TagLib to the login.jsp page with a specific identifier (URI – Uniform Resource Identifier) pointing at the CSRF Guard library, in which a prefix was defined as “csrf”, which we will use to modify the tag of the form that requires the input of the username and password.



This way the form is now guarded against CSRF attacks.

```
57     </ul>
58 </div>
59
60     <div id="content">
61         <csrf:form action="ValidateLogin" method="post">
62             <table>
63                 <tr><td>UserName: </td><td><input type="text" name="username" value="<%=username%>" /></td></tr>
64                 <tr><td>Password :</td><td><input type="password" name="password" value="<%=password%>" /></td></tr>
65                 <tr><td><input type="submit" name="Login" value="Login" /></td></tr>
66             </table>
67         </csrf:form>
68
69         <div id="footer"><h3><a href="http://www.donaldtrumpstinks.com/">Trump Web Design</a></h3></div>
70     </div>
```