Software Security

Module 10 - Authentication & Session Management

CMPE279
Software Security Technologies
San Jose State University

HTTP

Authentication

https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Authentication.html

HTTP Authentication

HTTP provides two schemes for authenticating clients: *Basic Access Authentication* and *Digest Access Authentication*. The specification is given in "RFC 2617 HTTP Authentication: Basic and Digest Access Authentication". It is important to stress that these schemes merely provide a mean for the client to send in his username/password for authentication. The schemes do not ensure message confidentially (i.e., message encryption), message integrity (man-in-the-middle attack), and non-repudiation. For high-risk and high-security systems which require these guarantees, you have to turn to SSL (Secure Socket Layout). HTTP with SSL (or HTTPS) will be discussed in the later chapter.

Basic Authentication

Basic Access Authentication scheme was introduced since HTTP/1.0. It is a simple scheme, which uses username/password to authenticate clients.

The client sends his username and password to the server. The server looks up the password file and decide whether the client is authorized to access the requested resource. The password file can be a simple text file, or a database (which greatly improves the retrieval and matching process for large user base). Users can be grouped into *groups*. Access can then be controlled at the group level. To improve on the flexibility, the protected space is divided into *realm*. Each username/password pair is valid for one particular realm.

CAUTION: The Basic Access Authentication scheme is not considered secure nor safe. This is because the username, password, and messages are all sent in clear text (i.e., not encrypted). Eavesdroppers or network snifters can easily pick up your username, password, and the messages. You should be extremely cautious when giving your username and password to a server that uses basic authentication scheme (if you use the same password to access your bank account!?). Use a dedicated "low-risk" password for this site instead. (The morale of this story is you should use different passwords for different purposes, although it could be hard to manage.)


Browser
Server
GET /basic_auth/test.html HTTP/1.1
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
(blank line)
HTTP/1.1 401 Authorization Required
Date: Tue, 20 Oct 2009 07:25:42 GMT
Server: Apache/2.2.14 (Win32)
Authentication Required
A username and password are being requested by http://127.0.0.1:8000. The site says: "Private"
User Name:
Password:
OK
Cancel
GET /basic_auth/test.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
Authorization: Basic YWxpY2U6YWxpY2U=
HTTP/1.1 200 OK
Date: Tue, 20 Oct 2009 07:37:12 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "f00000003e4f9-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Content-Type: text/html
(body omitted)

Security Exposure

In basic access authentication, the credential client sends to the server is not encrypted but simply *Base64-encoded* (refer to RFC2045 for specification on Base64 encoding). If you look up the Base64 table (or use WinZip), you will see the username/password in the `Authorization` header in clear text.

The most serious flaw in basic access authentication is that it transmits the password in clear text over the network. Hence, this password should not be the same as the one you use to access your bank account. Furthermore, once authenticated, the message is also sent in clear text. The username, password, and the message are subjected to network sniffing.

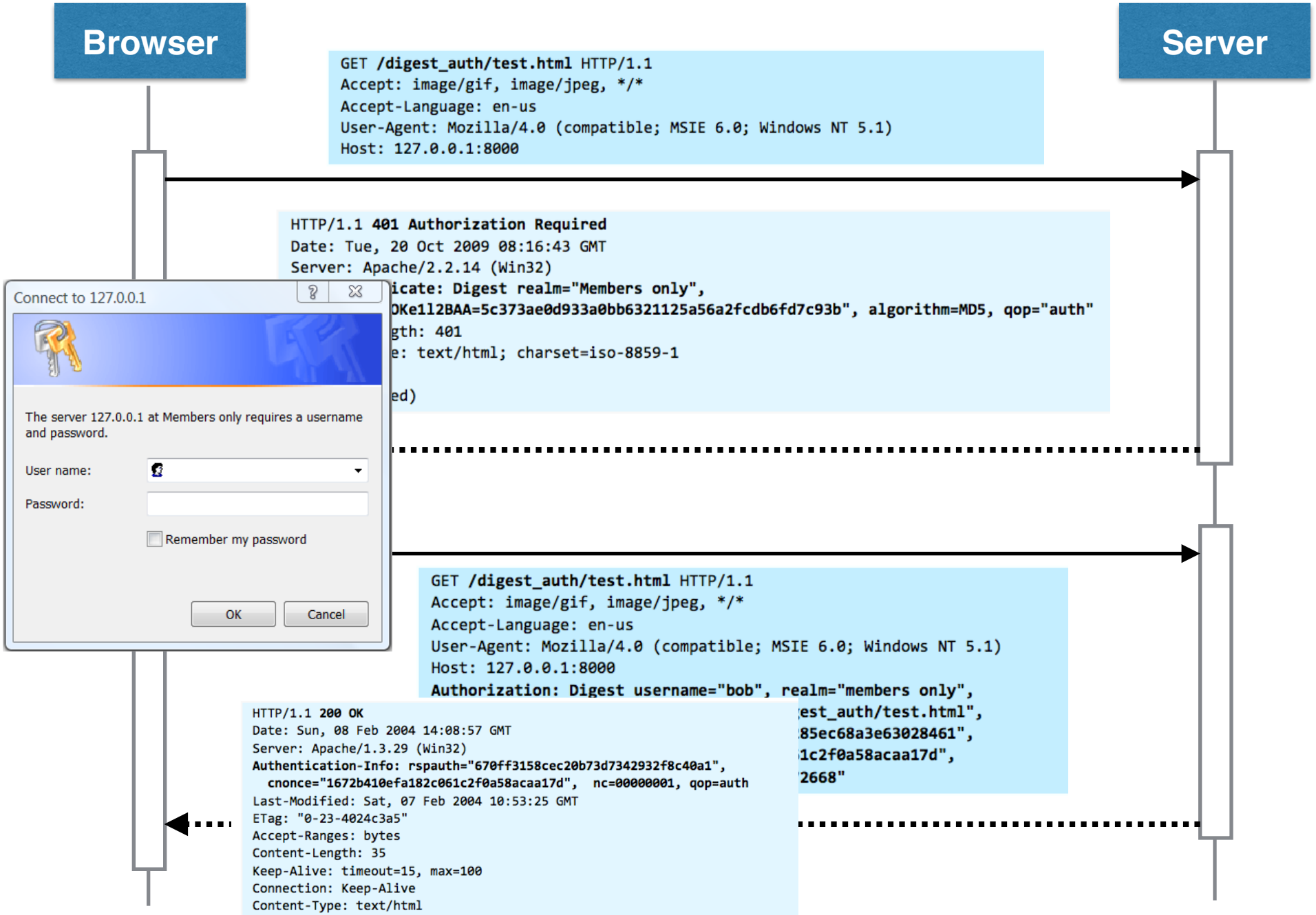
HTTP Digest Auth Protocol

The details of the authentication steps involved in the digest scheme are as follows:

1. The client requests for a web resource protected by digest authentication scheme, via an HTTP GET request.
2. The server returns a "401 Authentication Required" response, specifying the authentication scheme, the realm, and a nonce, in the "`WWW-Authenticate`" response header.
3. The client combines the password, nonce, HTTP method and URI; computes the digest; and sends this digest back to the server. An example of computation is as shown:

```
MD5(MD5(<password> + ":" + <nouce> + ":" + MD5(<method> + ";" + <uri>)
```

4. The server independently generates the hash and verifies with the hash received. It sends back the resource requested (with status "200 OK") if the two hash values match. Otherwise, a "401 Authentication Required" will be sent, and the authentication process repeats (or until the client cancels the request).


Browser
Server
GET /digest_auth/test.html HTTP/1.1
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
HTTP/1.1 401 Authorization Required
Date: Tue, 20 Oct 2009 08:16:43 GMT
Server: Apache/2.2.14 (Win32)
icate: Digest realm="Members only",
OKe1l2BAA=5c373ae0d933a0bb6321125a56a2fcdb6fd7c93b", algorithm=MD5, qop="auth"
gth: 401
e: text/html; charset=iso-8859-1
ed)
Connect to 127.0.0.1
The server 127.0.0.1 at Members only requires a username and password.
User name:
Password:
Remember my password
OK
Cancel
GET /digest_auth/test.html HTTP/1.1
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Authorization: Digest username="bob", realm="members only",
est_auth/test.html",
85ec68a3e63028461",
1c2f0a58acaa17d",
2668"
HTTP/1.1 200 OK
Date: Sun, 08 Feb 2004 14:08:57 GMT
Server: Apache/1.3.29 (Win32)
Authentication-Info: rspauth="670ff3158cec20b73d7342932f8c40a1",
cnonce="1672b410efa182c061c2f0a58acaa17d", nc=00000001, qop=auth
Last-Modified: Sat, 07 Feb 2004 10:53:25 GMT
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
(body omitted)

Security Considerations

Basic access authentication scheme has the following flaws:

- Password is sent in cleartext.
- Server must keep the username/password in the database. If the password file is compromise, all users will be affected (although most likely the password file is encrypted).
- No message integrity.

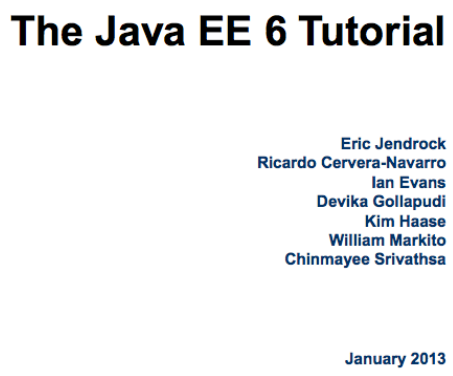
Digest scheme overcomes these flaws. However, it does not assure message confidentiality and non-repudiation. It merely provides a more secure and safe way for clients to submit their credential. It is not as secure as Kerberos, client-side private key, or SSL, but much better than telnet, ftp and of course, basic authentication scheme.

You should certainly replace the basic scheme with the digest scheme, unless password security is no of your concern.

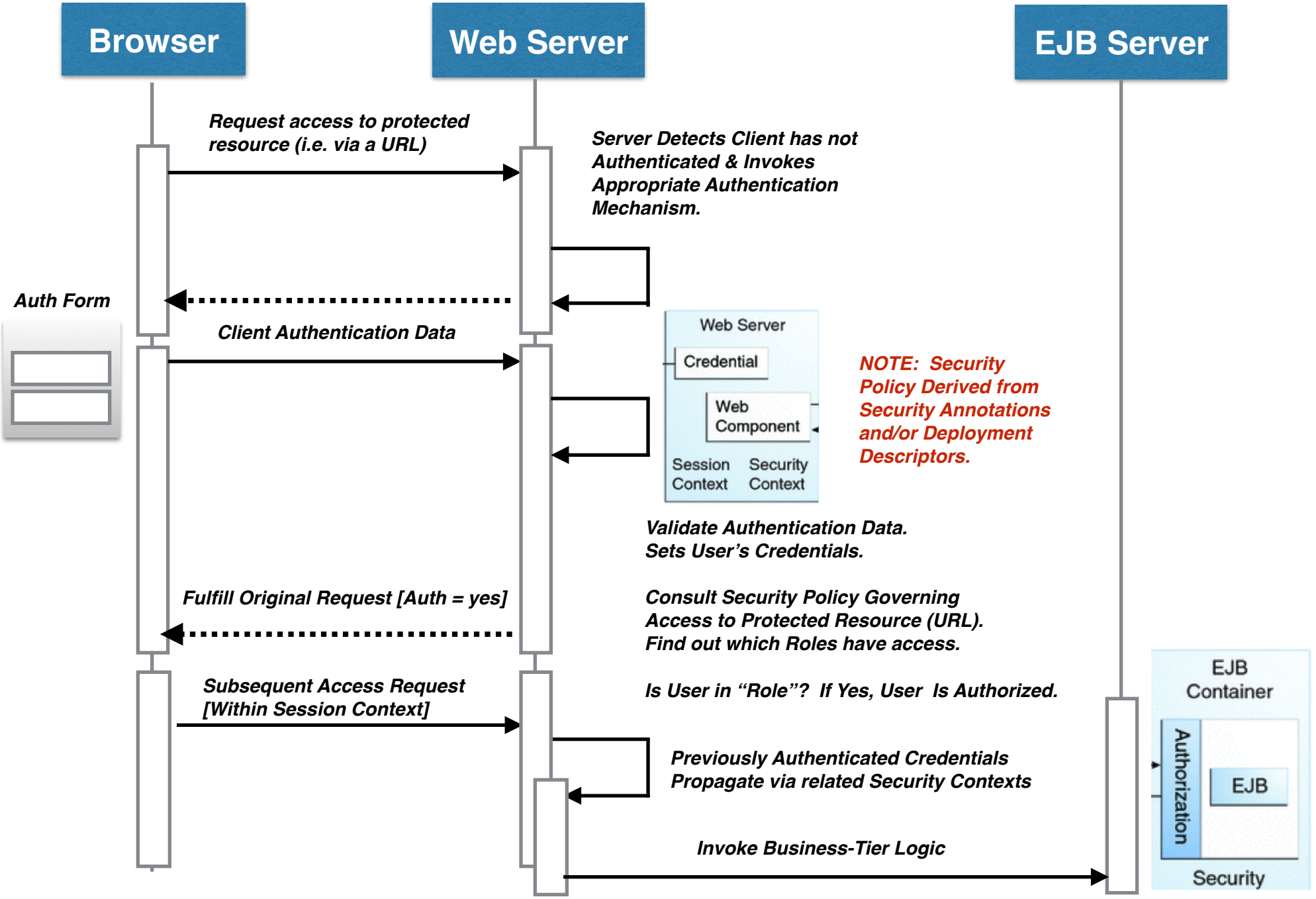
One final note: by modern cryptographic standards, the MD5 digest algorithm is considered weak. (Need to verify!)

J2EE Security

Example Walkthrough



http://docs.oracle.com/javaee/6/tutorial/doc/


Browser
Web Server
EJB Server
Request access to protected resource (i.e. via a URL)
Server Detects Client has not Authenticated & Invokes Appropriate Authentication Mechanism.
Auth Form
Client Authentication Data
Web Server
Credential
Web Component
Session Context
Security Context
NOTE: Security Policy Derived from Security Annotations and/or Deployment Descriptors.
Validate Authentication Data.
Sets User's Credentials.
Fulfill Original Request [Auth = yes]
Consult Security Policy Governing Access to Protected Resource (URL).
Find out which Roles have access.
Subsequent Access Request [Within Session Context]
Is User in "Role"? If Yes, User Is Authorized.
Previously Authenticated Credentials Propagate via related Security Contexts
Invoke Business-Tier Logic
EJB Container
Authorization
EJB
Security

The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Devika Gollapudi
Kim Haase
William Markito
Chinmayee Srivathsa

January 2013

J2EE Security

Securing Web Applications

Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

Quick Glossary

A **user** is the entity that wants to access a resource in our system. The user can either be another program or it can be a human user. A user is usually a member of one or more groups. A user may have an anonymous principal assigned by default, and can also have a unique security principal.

A **group** is an abstract name for an arbitrary number of users with common characteristics, which usually leads them to have a common set of access permissions.

A **security realm** is the abstraction on top of the storage that our users and groups are stored in. For example, we can have a JDBC security realm or an LDAP security realm, in which the first one is the abstraction on top of the tables and columns in the database holding our users' information, whereas, in the second one we are referring to the abstraction over the LDAP directory server containing our users, and their group associations.

A **role** is an application level concept that we use, to design the application security levels and permissions. A role in the application can map one or more groups in the security realm. This is how the application security is independent of where the users are stored, or how the users and groups are associated together in the environment in which we will deploy our application.

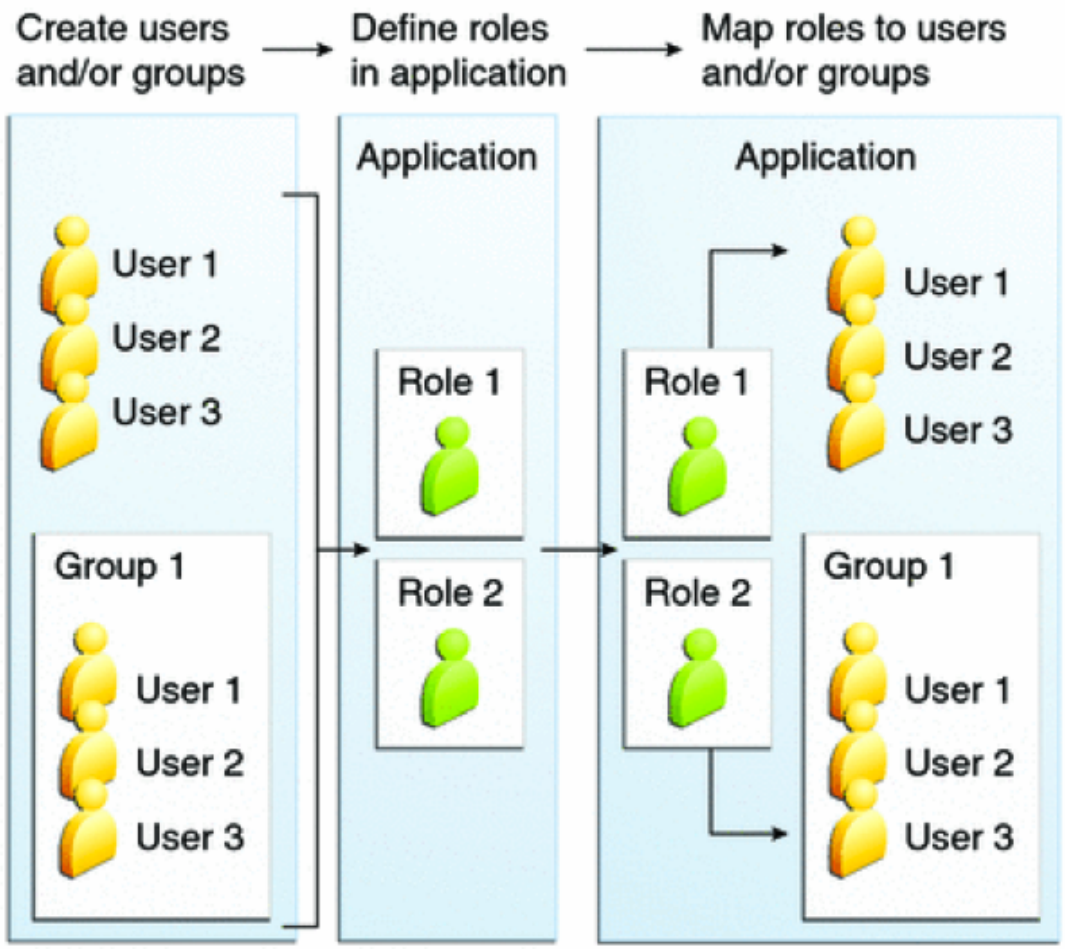
A **principal** is an identity that can be authenticated using one or more credentials, which are part of the principal.

A **credential** contains or references information used to authenticate a principal. The combination of a username and password is a sample of credentials.

Realms, Users, Groups & Roles

A **realm** is a security policy domain defined for a web or application server. A realm contains a collection of users, who may or may not be assigned to a group. Managing users on the GlassFish Server is discussed in Managing Users and Groups on the GlassFish Server.

Figure 39-6 Mapping Roles to Users and Groups



File, Certificate & Admin Realms

What Is a Realm?

The protected resources on a server can be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database containing a collection of users and groups. A realm is a complete database of users and groups identified as valid users of one or more applications and controlled by the same authentication policy.

The Java EE server authentication service can govern users in multiple realms. The `file`, `admin-realm`, and `certificate` realms come preconfigured for the GlassFish Server.

In the `file` realm, the server stores user credentials locally in a file named `keyfile`. You can use the Administration Console to manage users in the `file` realm. When using the `file` realm, the server authentication service verifies user identity by checking the `file` realm. This realm is used for the authentication of all clients except for web browser clients that use HTTPS and certificates.

In the `certificate` realm, the server stores user credentials in a certificate database. When using the `certificate` realm, the server uses certificates with HTTPS to authenticate web clients. To verify the identity of a user in the `certificate` realm, the authentication service verifies an X.509 certificate. For step-by-step instructions for creating this type of certificate, see Working with Digital Certificates. The common name field of the X.509 certificate is used as the principal name.

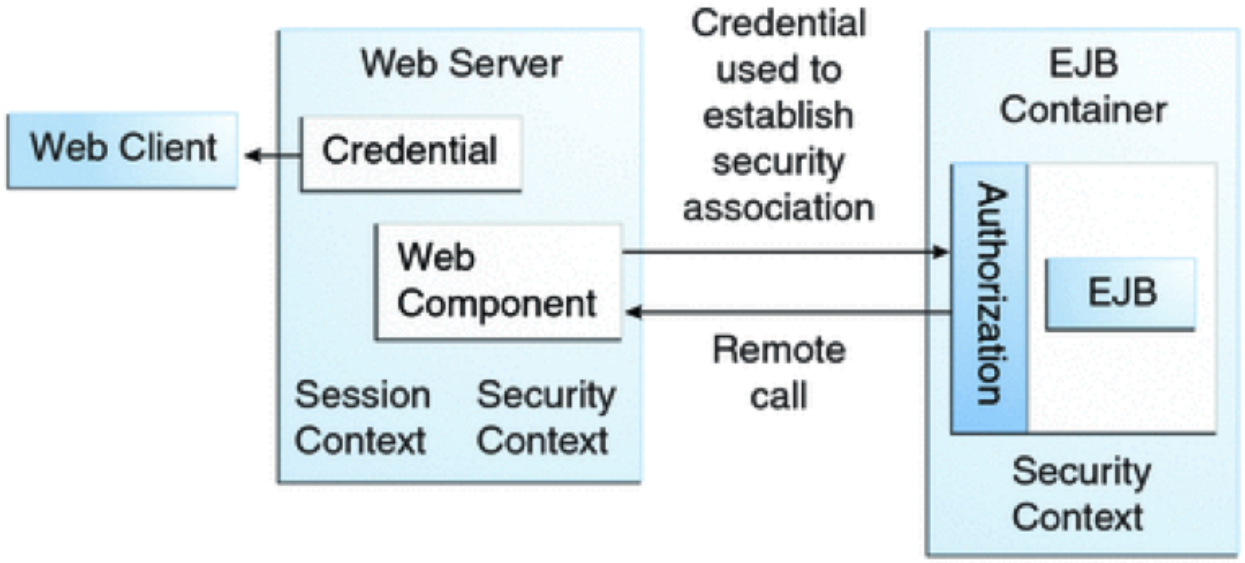
The `admin-realm` is also a `file` realm and stores administrator user credentials locally in a file named `admin-keyfile`. You can use the Administration Console to manage users in this realm in the same way you manage users in the `file` realm. For more information, see Managing Users and Groups on the GlassFish Server.

Java EE Containers

Security and Java EE containers

When a user is authenticated in one container, for example, the Web container, and there is a call from the Web container to the EJB container, the user's principal will propagate to the EJB container, and the EJB container will use the same principal for access control.

The same identity propagation happens when we are using the Application Client container, hosting an application client that interacts with the EJB container. The authentication happens in the Application Client container, and the identity is propagated to the EJB container when an invocation is performed.



Declarative vs. Programmatic Security

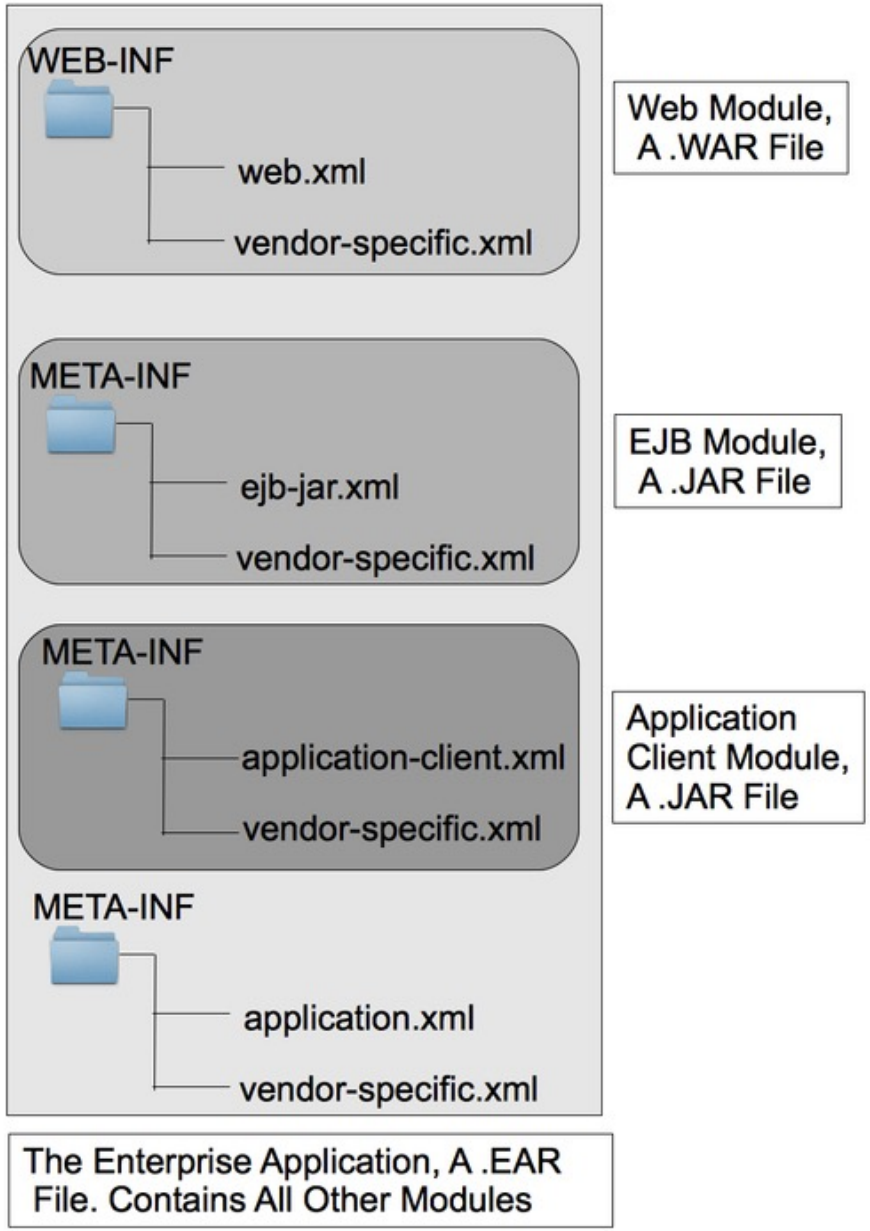
- **Declarative security** expresses an application component's security requirements by using either deployment descriptors or annotations.

  A deployment descriptor is an XML file that is external to the application and that expresses an application's security structure, including security roles, access control, and authentication requirements. For more information about deployment descriptors, read Using Deployment Descriptors for Declarative Security.

  Annotations, also called metadata, are used to specify information about security within a class file. When the application is deployed, this information can be either used by or overridden by the application deployment descriptor. Annotations save you from having to write declarative information inside XML descriptors. Instead, you simply put annotations on the code, and the required information gets generated. For this tutorial, annotations are used for securing applications wherever possible. For more information about annotations, see Using Annotations to Specify Security Information.

- **Programmatic security** is embedded in an application and is used to make security decisions. Programmatic security is useful when declarative security alone is not sufficient to express the security model of an application. For more information about programmatic security, read Using Programmatic Security.

Java EE deployment descriptors



Some of the security configurations are made in the ***standard deployment*** *descriptors and are the same for all vendors.*

Others are ***open for the vendors to specify*** *how administrators and developers should use them*

The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Devika Gollapudi
Kim Haase
William Markito
Chinmayee Srivathsa

January 2013

J2EE Security

Authentication

Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

Five Options for Authentication in JavaEE

- HTTP Basic Authentication
- HTTPS Client Authentication (i.e.Client Cert)
- Transport Security (i.e. SSL/HTTPS)
- Digest Authentication
- Form Based Authentication

HTTP Basic Authentication

Browser Presents a Standard Dialog for Login & Password

Easy to Implement

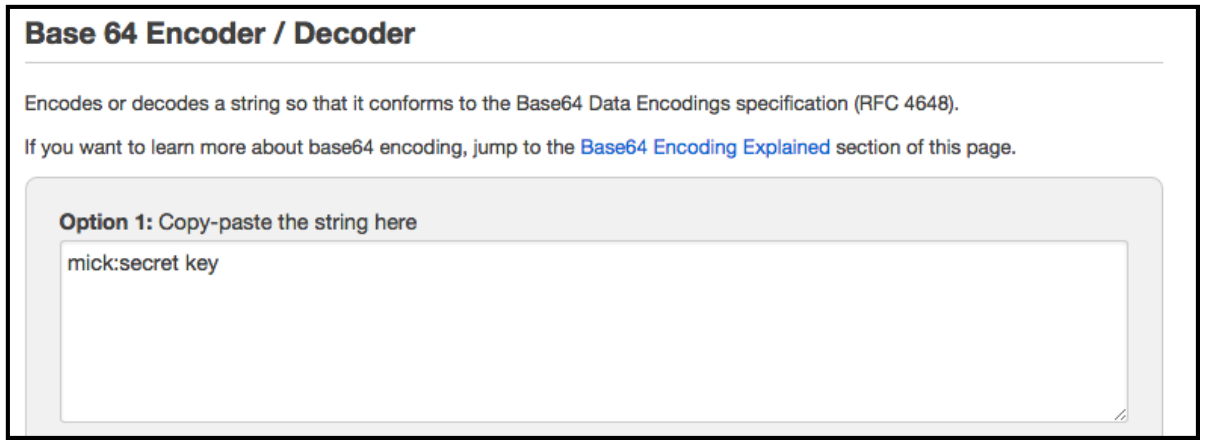
Credentials Sent in "Clear Text".

Need HTTPS.

Credentials are Concatenated and then Base64 Encoded

---

http://www.freeformatter.com/base64-encoder.html

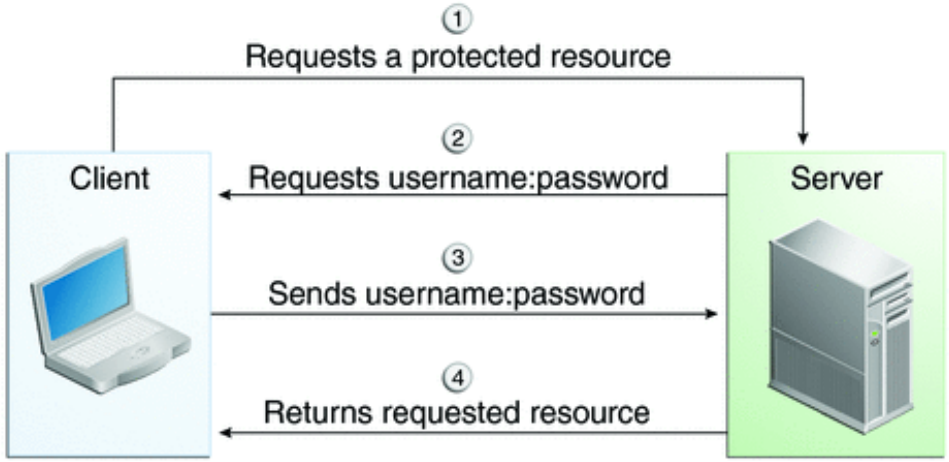


HTTP Basic Authentication

Basic authentication is the default when you do not specify an authentication mechanism.

When basic authentication is used, the following actions occur:

1. A client requests access to a protected resource.
2. The web server returns a dialog box that requests the user name and password.
3. The client submits the user name and password to the server.
4. The server authenticates the user in the specified realm and, if successful, returns the requested resource.



http://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpie/index.html

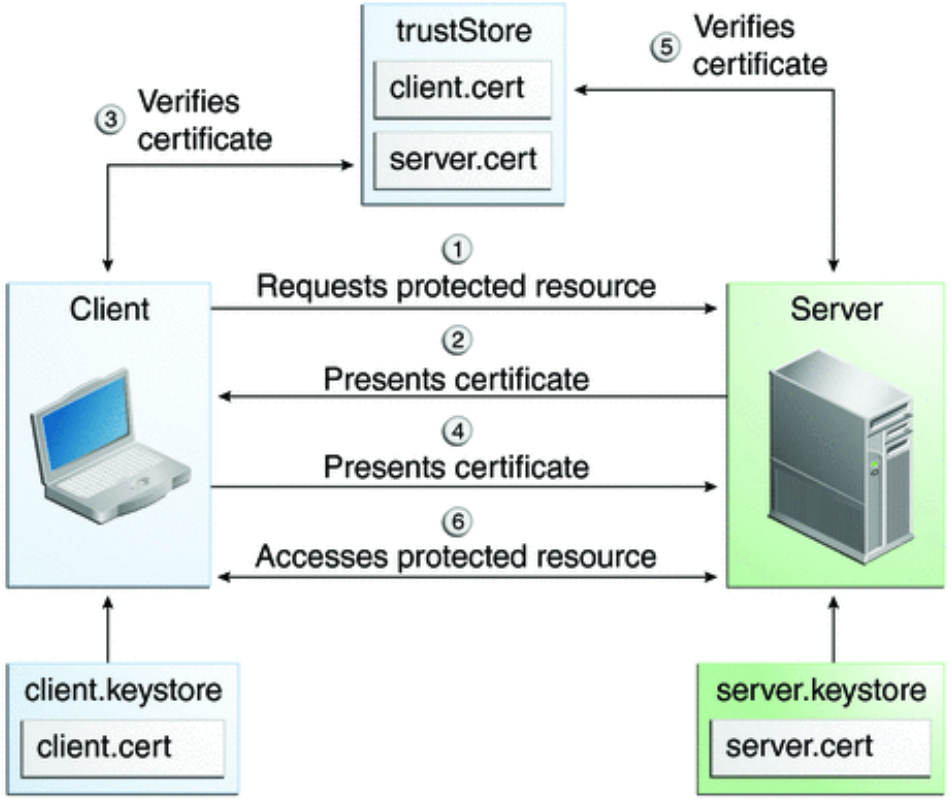
HTTPS Client Authentication (i.e.Client Cert)

Both Client and Server User Certs to Identify Themselves

Client must be able to validate Server's Cert.

Server must be able to validate Client's Cert.

Very Secured, but hard to Implement.



http://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpie/index.html

Transport Security (i.e. SSL/HTTPS)

Defined in "web.xml"
In addition to other
Methods

```
<?xml version="1.0" encoding="UTF-8"?>
<security-constraint>
   <web-resource-collection>
      <web-resource-name>customer</web-resource-name>
      <url-pattern>/customers/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>customer</role-name>
   </auth-constraint>
   <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

Options:

1. CONFIDENTIAL
2. INTEGRAL
3. NONE

*Specify **CONFIDENTIAL** when the application requires that data be transmitted so as to prevent other entities from observing the contents of the transmission.*

*Specify **INTEGRAL** when the application requires that the data be sent between client and server in such a way that it cannot be changed in transit.*

*Specify **NONE** to indicate that the container must accept the constrained requests on any connection, including an unprotected one.*

http://docs.oracle.com/cd/E19798-01/821-1841/bncbk/index.html

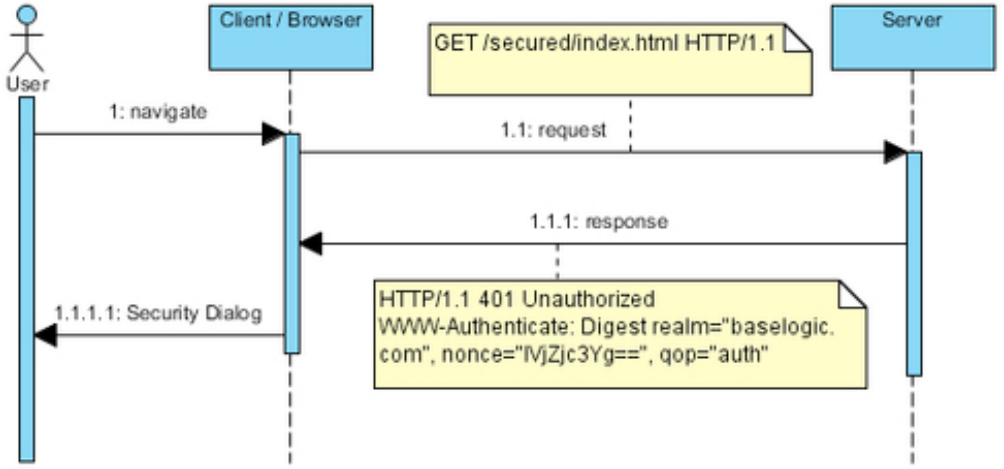
Digest Authentication

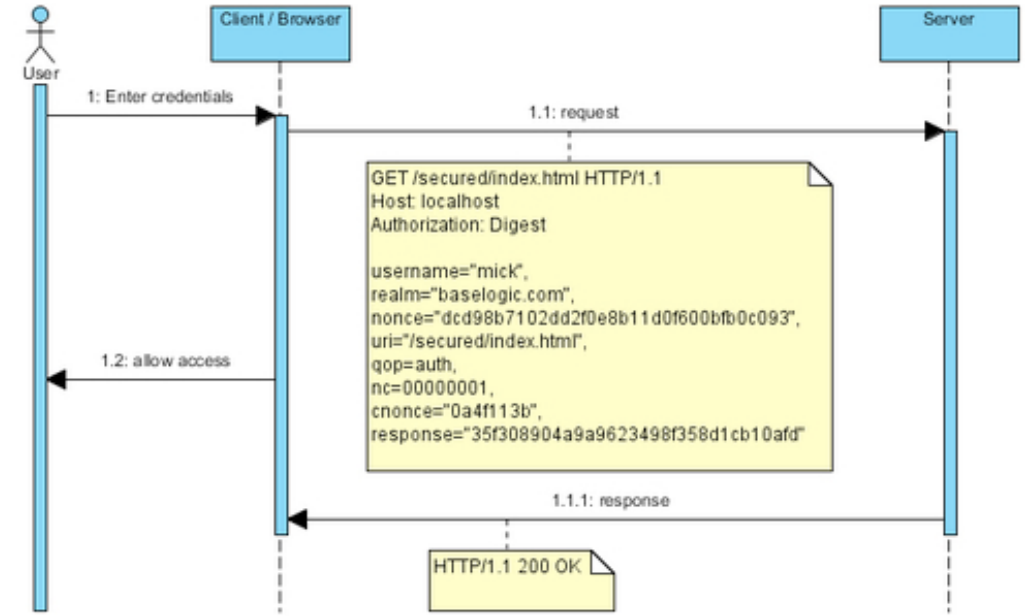
Similar to Basic Auth

But a Digest of the Password is Transmitted Instead

Client sends a one-way Cryptographic hash of the Password

Requires that clear-text Password equivalents be Available to the server For Validation





Form Based Authentication

Custom HTML Form Provided by App (Defined in web.xml)

Like Basic Auth, Credentials sent in Clear Text

Sample "web.xml" Config

Sample "Form" Post to "j_security_check"

```
<?xml version="1.0" encoding="UTF-8"?>
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>file</realm-name>
   <form-login-config>
      <form-login-page>/auth/login.html</form-login-page>
      <form-error-page>/auth/loginError.html</form-error-page>
   </form-login-config>
</login-config>
```

```
<html>
   <head>
      <title></title>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8
         </head>
         <body>
         <form method="POST" action="j_security_check">
      Username: <input type="text" name="j_username"><br />
      Password: <input type="password" name="j_password"><br />
      <br />
      <input type="submit" value="Login">
      <input type="reset" value="Reset">
      </form>
      </body>
</html>
```

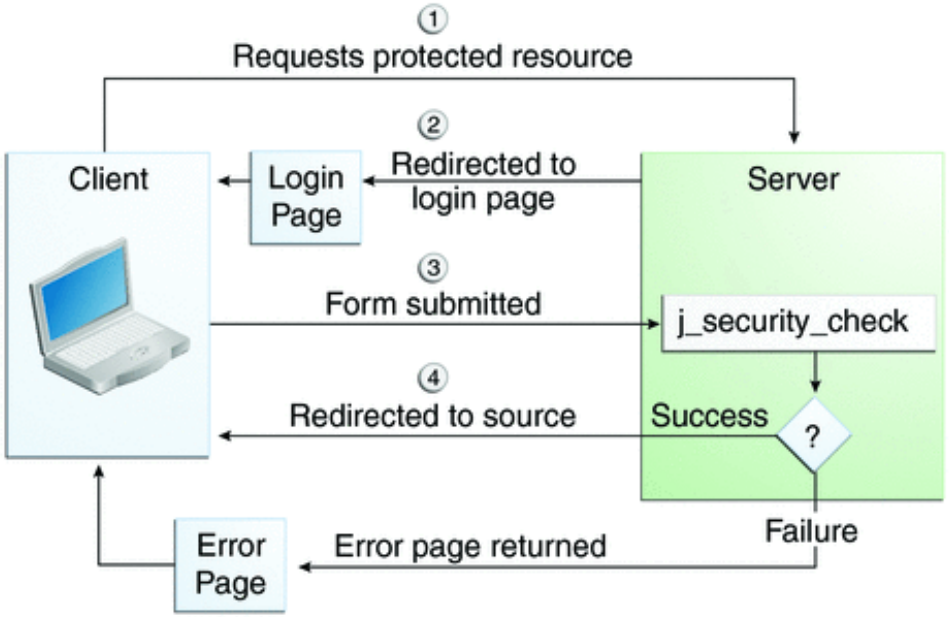
Form-Based Authentication

Form-based authentication allows the developer to control the look and feel of the login authentication screens by customizing the login screen and error pages that an HTTP browser presents to the end user. When form-based authentication is declared, the following actions occur.

1. A client requests access to a protected resource.
2. If the client is unauthenticated, the server redirects the client to a login page.
3. The client submits the login form to the server.
4. The server attempts to authenticate the user.

    A. If authentication succeeds, the authenticated user's principal is checked to ensure that it is in a role that is authorized to access the resource. If the user is authorized, the server redirects the client to the resource by using the stored URL path.

    B. If authentication fails, the client is forwarded or redirected to an error page.




```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="password" name="j_password">
</form>
```

http://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpie/index.html

The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Devika Gollapudi
Kim Haase
William Markito
Chinmayee Srivathsa

January 2013

J2EE Security

Authorization

Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

Configuration Authorization in web.xml

Two Steps:

1. **Specify the restricted resources and the roles that are able to access these resources**

2. **Mapping these roles to actual user groups in the security realm**

```
<?xml version="1.0" encoding="UTF-8"?>
<security-constraint>
   <display-name>You should be a manager</display-name>
   <web-resource-collection>
      <web-resource-name>Inch</web-resource-name>
      <description />
      <url-pattern>/jsp/toInch.jsp</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
   </web-resource-collection>
   <auth-constraint>
      <description />
      <role-name>manager_role</role-name>
   </auth-constraint>
</security-constraint>
```

Different application servers use different ways to map the roles to groups and principals. Here, we will show how it can be done in the GlassFish application server. For the GlassFish application server, we have `glassfish-web.xml` for GlassFish 3.1 and above, or the `sun-web.xml` file for older versions, to use for the vendor-specific configurations. The file is located in the same directory as the `web.xml` file, which is `WEB-INF/`:

```
<security-role-mapping>
<role-name>manager_role</role-name>
<group-name>manager_group</group-name>
<principal-name>tom</principal-name>
</security-role-mapping>
```

Configuration Authorization in web.xml

Element	Description
`security-constraint`	The top element we use to define a restriction over a set of resources for specified HTTP methods in our web applications.
`web-resource-collection`	The element used to specify the resource URLs or URL patterns as well as the set of HTTP methods we want to protect the URL patterns from.
`url-pattern`	Specifies the URLs or the URL patterns we want to put under protection of this constraint. We can have as many `url-pattern` elements for `web-resource-collection` as we need.
`http-method`	The HTTP methods we want the resource collection to be protected from. If no http-method elements are defined, the collection will be protected from all methods. It is recommended that we protect the resource from all methods unless we don't need to.
`auth-constraint`	This element specifies the list of roles that are allowed to access the resource collection.
`role-name`	Name of the roles we want to grant access to the collection. We can have as many `role-name` elements as we need.

Securing EJB's (ejb-jar.xml)

Enforcing Access Control

Only Users with Roles:

1. manager_role
2. hr_role"

Can Update Employee Salaries

```xml
<?xml version="1.0" encoding="UTF-8"?>
<assembly-descriptor>
   <method-permission>
      <role-name>manager_role</role-name>
      <role-name>hr_role</role-name>
      <method>
         <ejb-name>Employee</ejb-name>
         <method-name>updateSalary</method-name>
      </method>
      <method>
         <ejb-name>Employee</ejb-name>
         <method-name>getSalary</method-name>
      </method>
   </method-permission>
</assembly-descriptor>
```

Securing EJB's (ejb-jar.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<assembly-descriptor>
   <method-permission>
      <role-name>manager_role</role-name>
      <role-name>hr_role</role-name>
      <method>
         <ejb-name>Employee</ejb-name>
         <method-name>updateSalary</method-name>
      </method>
      <method>
         <ejb-name>Employee</ejb-name>
         <method-name>getSalary</method-name>
      </method>
   </method-permission>
</assembly-descriptor>
```

Element	Description
`method-permission`	The parent node, which contains permissions for one or more methods. Methods will have the same access level.
`role-name`	We can have as many roles for each method permission as we require.
`method`	Parent node, which contains the method we want to define its permission. We can have as many method elements as we require.
`ejb-name`	Name of the EJB.
`method-name`	Method name.

Securing EJB's (ejb-jar.xml)

```
<assembly-descriptor>
<description/>
<role-name>manager_role</role-name>
</assembly-descriptor>
```

Role Mappings

```
<security-role-mapping>
<role-name>manager_role</role-name>
<group-name>manager_group</group-name>
<principal-name>tom</principal-name>
</security-role-mapping>
<security-role-mapping>
<role-name>hr_role</role-name>
<group-name>hr_group</group-name>
</security-role-mapping>
```

Run As "Internal" Role

```
<enterprise-beans>
<session>
<ejb-name>ConversionBean</ejb-name>
<security-identity>
<!--<use-caller-identity/>-->
<run-as>internal_role</run-as>
</security-identity>
</session>
</enterprise-beans>
```

Internal roles in application (i.e. not user roles) used for for outgoing calls of an EJB.

Used to ensure that the outgoing calls from our EJB have the proper role assigned, independent of the roles that are assigned to the current user

The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Devika Gollapudi
Kim Haase
William Markito
Chinmayee Srivathsa

January 2013

J2EE Security

Programmatic Security

Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

Security Annotations in Java EE

Annotation	Description
`@ServletSecurity`	This is used to enforce security over a Servlet. This annotation causes the container to apply the security enforcement on the URL pattern defined for this Servlet. It can optionally get `@HttpMethodConstraint` and `@HttpConstraint` as its parameters. The `@HttpMethodConstraint` element is an array specifying the HTTP methods' specific constraint, while `@HttpConstraint` specifies the protection for all HTTP methods that are not specified in the `@HttpMethodConstraint`.
`@DeclareRoles`	This annotation declares the roles that we want to use in our application. It is similar to the `security-role` element in the deployment descriptor.
`@RolesAllowed`	An array of roles allowed to access the annotated target. For example, if the target is an EJB, then all of the EJB methods will be available to the list of roles unless they are marked with `@RolesAllowed` with a different set of roles.
`@PermitAll`	Permits all roles to access the annotated target.
`@DenyAll`	If placed on a method, no one can access that method. In the case of class level annotation, all methods of annotated EJB are inaccessible to all roles, unless a method is annotated with a `@RolesAllowed` annotation with an array or roles.
`@RunAs`	Works similar to `run-as` or `security-identity` elements discussed in previous recipes.

Security Annotations in Java EE

Annotation	Target kind	Target level
`@ServletSecurity`	Servlet	Class
`@DeclareRoles`	Servlet, EJB	Class
`@RolesAllowed`	EJB	Class, Method
`@PermitAll`	EJB	Class, Method
`@DenyAll`	EJB	Method
`@RunAs`	Servlet, EJB	Class

Annotation Example

```
@ServletSecurity(@HttpConstraint (rolesAllowed = {" employee", " manager"}))
```

```
<security-constraint>
<web-resource-collection>
<url-pattern>/mappedURL</url-pattern>
</web-resource-collection>
<auth-constraint>
<security-role-name>manager</security-role-name>
<security-role-name>employee</security-role-name>
</auth-constraint>
</security-constraint>
```

HTTP Servlet Request Methods

Method	Description
`void login(String username, String password)`	Authenticates the given username and password with the configured security realm for the application, and if successful, logs in the user. It is similar to the `BASIC` or `FORM` methods, but with the developer completely in control of the process.
`Void logout()`	Resets the current return value of `getUserPrincipal`, `getRemoteUser`, and `getAuthType` to null.
`boolean isUserInRole(String role)`	Checks whether the current user has the given role or not.
`String getAuthType()`	Returns the authentication type configured for the application.
`String getRemoteUser()`	If the user is authenticated, it returns the username otherwise it returns null.
`Principal getUserPrincipal()`	Returns a `java.security.Principal` object corresponding to the current user.
`String getScheme()`	Returns the URL schema, `HTTP` or `HTTPS`.

HTTP Servlet Request Methods

```
String user = request.getParameter("user");
String password = request.getParameter("password");
try {
request.login(user, password);
if (request.isUserInRole("manager ")) response.sendRedirect("/mgr/index.jsp");
else response.sendRedirect ("/emp/index.jsp");
}catch(ServletException ex) {
//Handling Exception
return;
}
```

The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Devika Gollapudi
Kim Haase
William Markito
Chinmayee Srivathsa

January 2013

J2EE Security

Web.xml Security Misconfigurations

Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

1) Custom Error Pages Not Configured

By default Java web applications display detailed error messages that disclose the server version and detailed stack trace information that can, in some situations, wind up displaying snippets of Java code. This information is a boon to hackers who are looking for as much information about their victims as possible. Fortunately it's very easy to configure web.xml to display custom error pages. Using the following configuration a nice error page will be displayed whenever the application responds with an HTTP 500 error. You can add additional entries for other HTTP status codes as well.

```
<error-page>
<error-code>500</error-code>
<location>/path/to/error.jsp</location>
</error-page>
```

Additionally, web.xml needs to be configured to prevent detailed stack traces from being displayed by specifying the <exception-type> shown below. Since Throwable is the base class of all Exceptions and Errors in Java, this will ensure that stack traces aren't displayed by the server.

```
<error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/path/to/error.jsp</location>
</error-page>
```

However, your code can still show stack traces if you do something like this:

```
<%
try {
String s = null;
s.length();
} catch (Exception e) {
// don't do this!
e.printStackTrace(new PrintWriter(out));
}
%>
```

Remember to use appropriate logging in addition to properly configuring your web.xml file.

HTTP Methods Accidentally Allowed

2) Authentication & Authorization Bypass

The following code shows how to set up web-based access control so that everything in the "secure" directory should only be accessible to users with the "admin" role.

```
<security-constraint>
<web-resource-collection>
<web-resource-name>secure</web-resource-name>
<url-pattern>/secure/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>admin</role-name>
</auth-constraint>
</security-constraint>
```

From a common sense point of view, the <http-method> elements that specify GET and POST should indicate that *only* GET and POST requests are allowed. That is not the case, as any HTTP methods that are *not* explicitly listed are in fact allowed. Arshan Dabirsiaghi has a nice paper that summarizes this issue and shows how to use arbitrary HTTP verbs (like HEAD) and completely fake verbs (like "TEST" or "JUNK") which are not listed in the configuration to bypass web.xml authentication and authorization protections.

Fortunately, the solution is simple. Just remove all <http-method> elements from your web.xml and the configuration will be properly applied to all requests.

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

No Protection for Data in Transit

3) SSL Not Configured

SSL should be used to protect data in transit in all applications that use sensitive data. You can of course configure SSL on the web server but, once your app server is set up with the appropriate SSL keys, it's very easy to enable SSL at the web app level if you so choose.

```
1  <security-constraint>
2  ...
3  <user-data-constraint>
4  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
5  </user-data-constraint>
6  </security-constraint>
```

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

Don’t Transmit Cookie over HTTP

4) Not Using the Secure Flag

Many web sites use SSL for authentication but then either revert back to non-SSL for subsequent communication or have parts of the site that can still be accessed via non-SSL. This leaves the session coookie (i.e. JSESSIONID) vulnerable to session hijacking attacks. To prevent this, cookies can be created with the "Secure" flag, which ensures that the browser will never transmit the specified cookie over non-SSL.

In older versions of the Servlet specification there wasn't a standard way (although there were vendor specific ways) to define the the JSESSIONID as "Secure". Now in Servlet 3.0 the <cookie-config> element can be used to ensure that the JSESSIONID is not transmitted over plain old HTTP.

```
1 <session-config>
2 <cookie-config>
3 <secure>true</secure>
4 </cookie-config>
5 </session-config>
```

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

Don’t Let JavaScript Access Cookies

5) Not Using the HttpOnly Flag

Cookies can be created with the "HttpOnly" flag, which ensures that the cookie cannot be accessed via client side scripts. This helps mitigate some of the most common XSS attacks. Just like the "Secure" flag, older versions of the Servlet specification didn't provide a standard way to define the JSESSIONID as "HttpOnly". Now in Servlet 3.0 the element can be configured in web.xml as follows:

```
1 <session-config>
2 <cookie-config>
3 <http-only>true</http-only>
4 </cookie-config>
5 </session-config>
```

Aside from this new standard approach in Servlet 3.0, older versions of Tomcat allowed the HttpOnly flag to be set with the vendor-specific "useHttpOnly" attribute for the <Context> in server.xml. This attribute was disabled by default in Tomcat 5.5 and 6. But now, in Tomcat 7, the "useHttpOnly" attribute is enabled by default. So, even if you configure web.xml to be <http-only>false</http-only> in Tomcat 7, your JSESSIONID will still be HttpOnly unless you change the default behaviour in server.xml as well.

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

Store JSESSIONID in Cookie (i.e. not URL)

6) Using URL Parameters for Session Tracking

The <tracking-mode> element in the Servlet 3.0 specification allows you to define whether the JSESSIONID should be stored in a cookie or in a URL parameter. If the session id is stored in a URL parameter it could be inadvertently saved in a number of locations including the browser history, proxy server logs, referrer logs, web logs, etc. Accidental disclosure of the session id makes the application more vulnerable to session hijacking attacks. Instead, make sure the JSESSIONID is stored in a cookie (and has the Secure flag set) using the following configuration:

```
1  <session-config>
2  <tracking-mode>COOKIE</tracking-mode>
3  </session-config>
```

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

Mitigate CSRF with Expired Sessions

7) Not Setting a Session Timeout

Users like long lived sessions because they are convenient. Hackers like long lived sessions because it gives them more time to conduct attacks like session hijacking and CSRF. Security vs usability will always be a dilemma. Once you know how long to keep your session alive you can configure the idle timeout as follows:

```
1  <session-config>
2  <session-timeout>15</session-timeout>
3  </session-config>
```

This examples has the session timing out after 15 minutes of inactivity. If the <session-timeout> element is not configured, the Servlet specification states that the container default should be used (for Tomcat this is 30 minutes). Specifying a number less than or equal to 0 means that the session will never expire. This is definitely not recommended, especially for high assurance applications.

The idle timeout can also be configured by using the setMaxInactiveInterval on the HttpSession class. Unlike the <session-timeout> element, however, this method takes the time in seconds.

In addition to the idle timeout, it would be nice if Java let you configure a hard timeout where the session would be destroyed after some set period of time even if the user was still actively using the application. Something like ASP.NET's slidingExpiration in web.config would be handy in some situations. There's no standard way to configure an absolute timeout but you could implement that logic in a ServletFilter.

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

The Java EE 6 Tutorial http://docs.oracle.com/javaee/6/tutorial/doc/

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Devika Gollapudi
Kim Haase
William Markito
Chinmayee Srivathsa

January 2013

J2EE Security

Authentication Examples
(with Netbeans and Glashfish Server)

Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications

▼ To Set Up Your System for Running the Security Examples

To set up your system for running the security examples, you need to configure a user database that the application can use for authenticating users. Before continuing, follow these steps.

1. Add an authorized user to the GlassFish Server. For the examples in this chapter and in Chapter 26, Getting Started Securing Enterprise Applications, add a user to the `file` realm of the GlassFish Server, and assign the user to the group `TutorialUser`:

    a. From the Administration Console, expand the Configuration node.

    b. Expand the Security node.

    c. Expand the Realms node.

    d. Select the File node.

    e. On the Edit Realm page, click Manage Users.

    f. On the File Users page, click New.

    g. In the User ID field, type a User ID.

    h. In the Group List field, type **`TutorialUser`**.

    i. In the New Password and Confirm New Password fields, type a password.

    j. Click OK.

    Be sure to write down the user name and password for the user you create so that you can use it for testing the example applications. Authentication is case sensitive for both the user name and password, so write down the user name and password exactly. This topic is discussed more in Managing Users and Groups on the GlassFish Server.

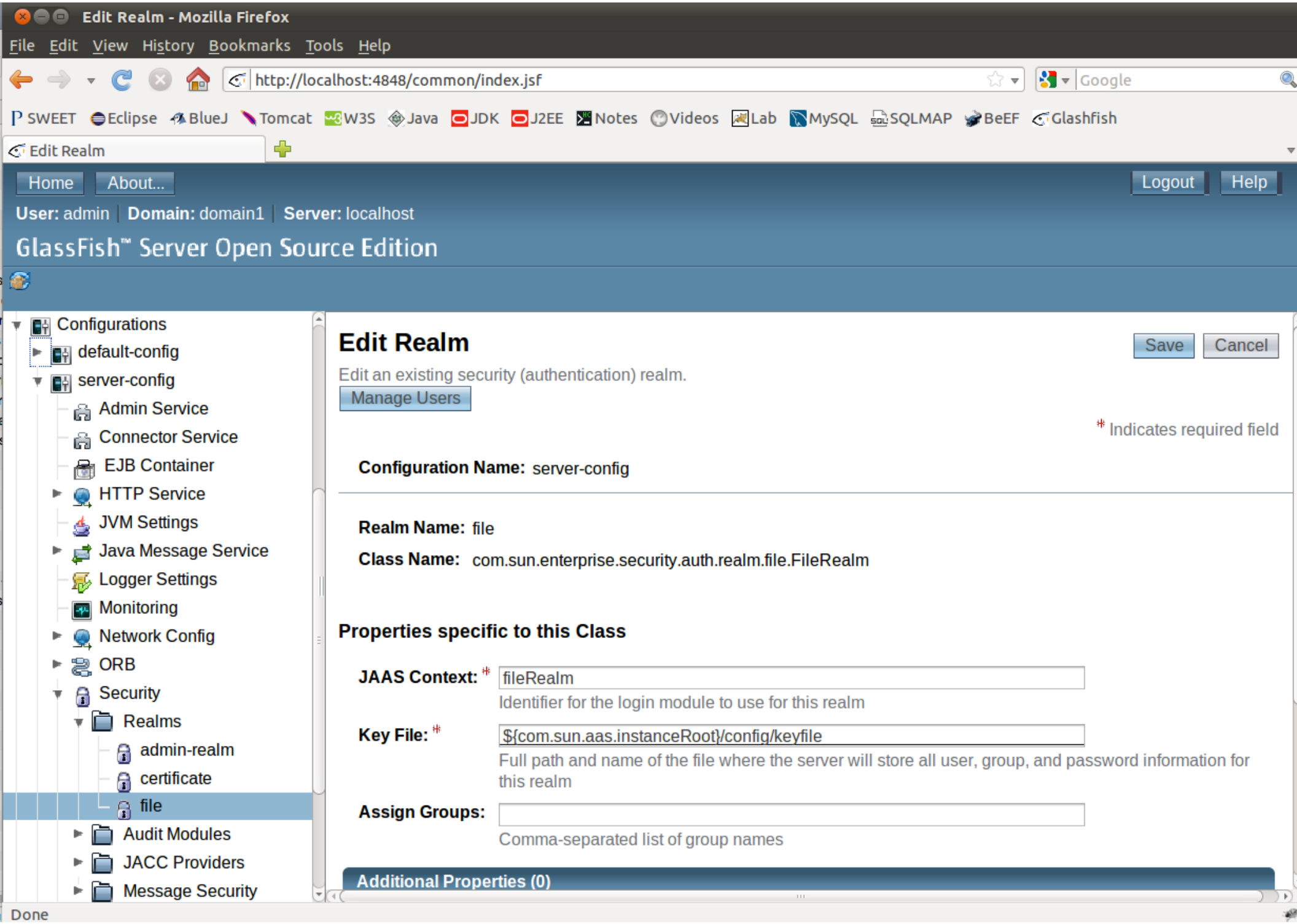
2. Set up Default Principal to Role Mapping on the GlassFish Server:

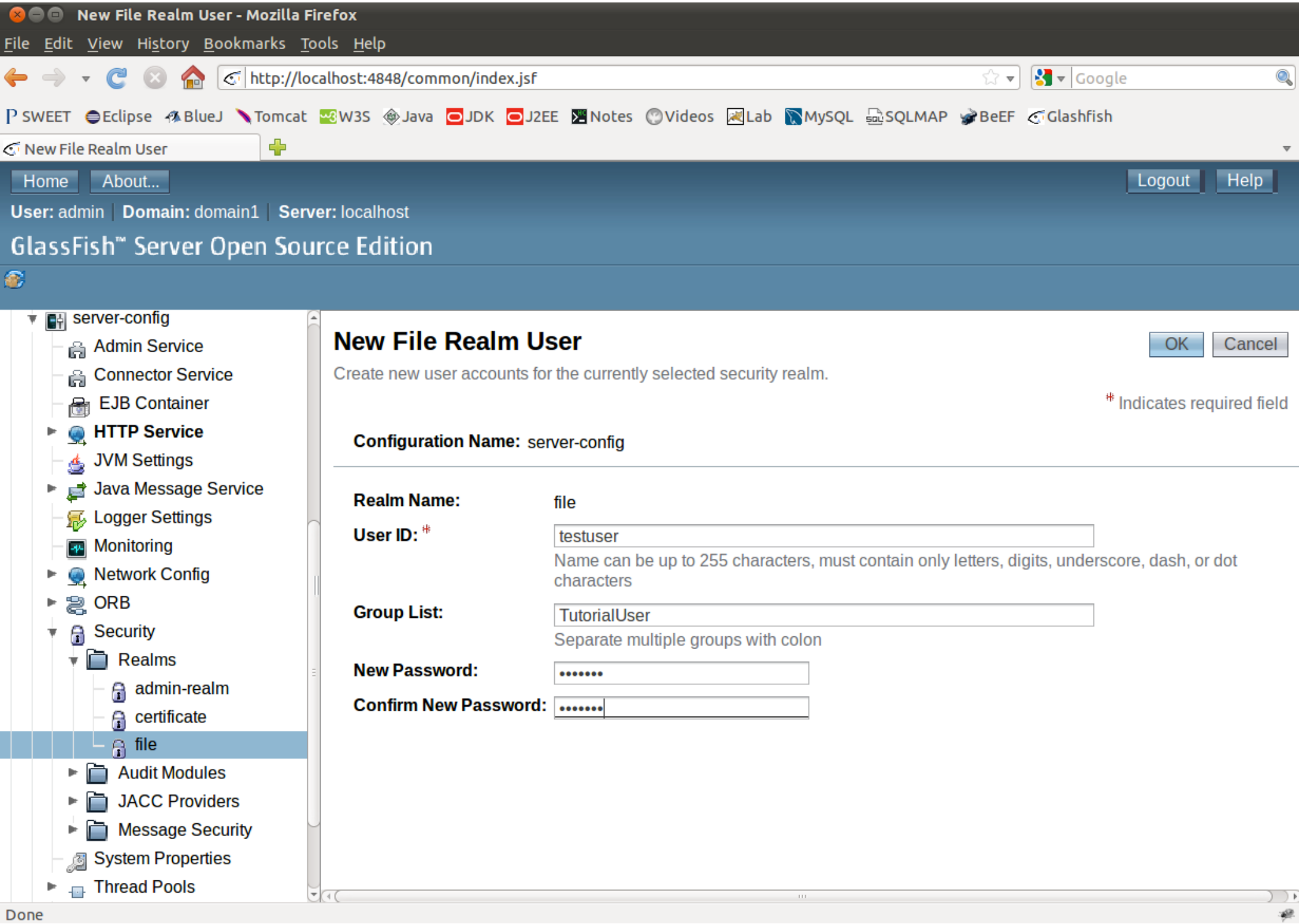
    a. From the Administration Console, expand the Configuration node.

    b. Select the Security node.

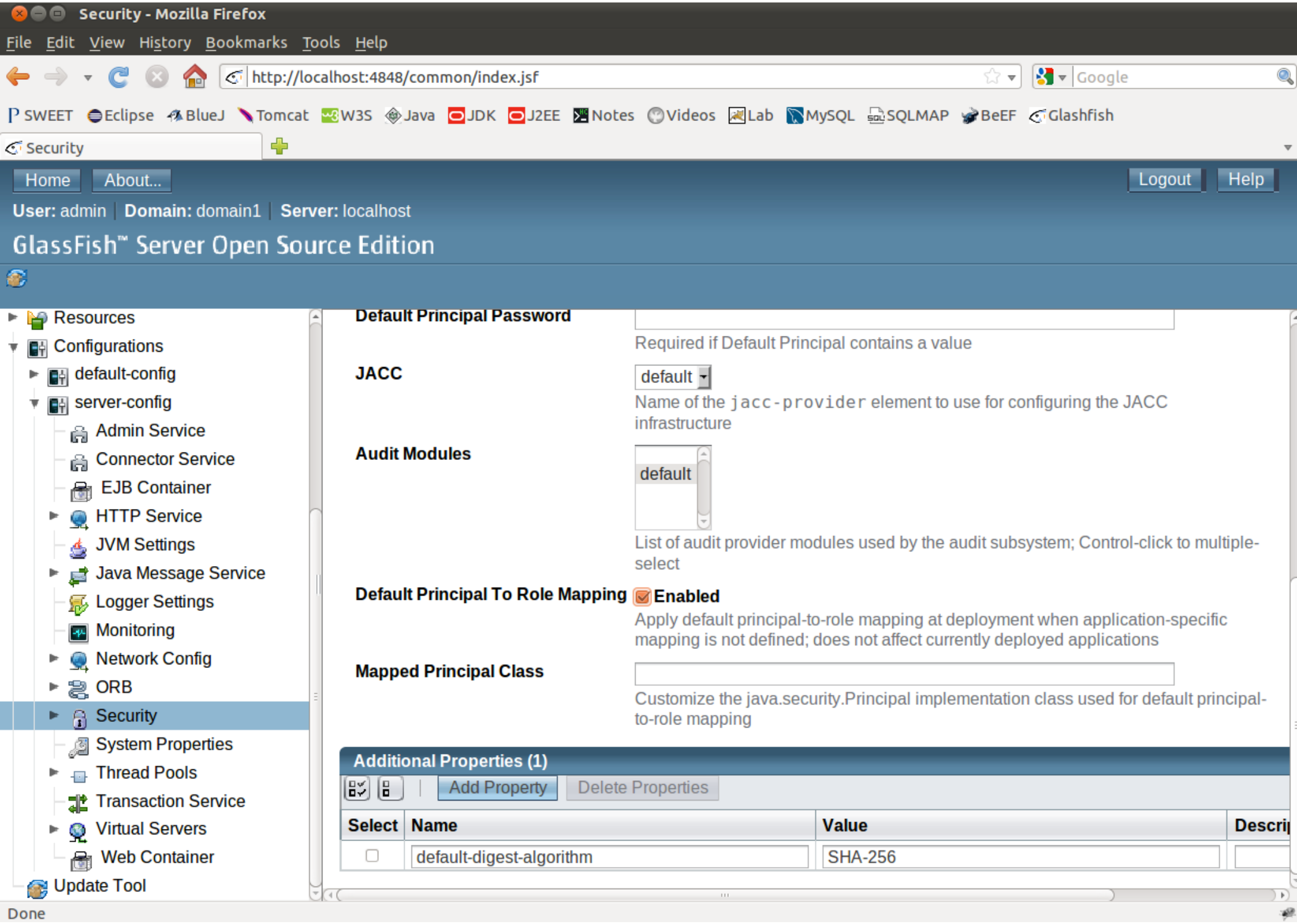
    c. Select the Default Principal to Role Mapping Enabled check box.

    d. Click Save.


Edit Realm - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:4848/common/index.jsf
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish
Edit Realm
Home About... Logout Help
User: admin | Domain: domain1 | Server: localhost
GlassFish™ Server Open Source Edition
Configurations
default-config
server-config
Admin Service
Connector Service
EJB Container
HTTP Service
JVM Settings
Java Message Service
Logger Settings
Monitoring
Network Config
ORB
Security
Realms
admin-realm
certificate
file
Audit Modules
JACC Providers
Message Security
Edit Realm
Save Cancel
Edit an existing security (authentication) realm.
Manage Users
* Indicates required field
Configuration Name: server-config
Realm Name: file
Class Name: com.sun.enterprise.security.auth.realm.file.FileRealm
Properties specific to this Class
JAAS Context: * fileRealm
Identifier for the login module to use for this realm
Key File: * ${com.sun.aas.instanceRoot}/config/keyfile
Full path and name of the file where the server will store all user, group, and password information for this realm
Assign Groups:
Comma-separated list of group names
Additional Properties (0)
Done



Login: testuser / welcome


Security - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:4848/common/index.jsf
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish
Security
Home About... Logout Help
User: admin | Domain: domain1 | Server: localhost
GlassFish™ Server Open Source Edition
Resources
Configurations
default-config
server-config
Admin Service
Connector Service
EJB Container
HTTP Service
JVM Settings
Java Message Service
Logger Settings
Monitoring
Network Config
ORB
Security
System Properties
Thread Pools
Transaction Service
Virtual Servers
Web Container
Update Tool
Default Principal Password
Required if Default Principal contains a value
JACC
default
Name of the jacc-provider element to use for configuring the JACC infrastructure
Audit Modules
default
List of audit provider modules used by the audit subsystem; Control-click to multiple-select
Default Principal To Role Mapping
Enabled
Apply default principal-to-role mapping at deployment when application-specific mapping is not defined; does not affect currently deployed applications
Mapped Principal Class
Customize the java.security.Principal implementation class used for default principal-to-role mapping
Additional Properties (1)
Add Property
Delete Properties
Select
Name
Value
Descri
default-digest-algorithm
SHA-256
Done

ORACLE

The Java EE 6 Tutorial

« **Previous**: To Set Up Your System for Running the Security Examples

Next: Example: Form-Based Authentication with a JavaServer Faces Application »

Example: Basic Authentication with a Servlet

This example explains how to use basic authentication with a servlet. With basic authentication of a servlet, the web browser presents a standard login dialog that is not customizable. When a user submits his or her name and password, the server determines whether the user name and password are those of an authorized user and sends the requested web resource if the user is authorized to view it.

In general, the following steps are necessary for adding basic authentication to an unsecured servlet, such as the ones described in Chapter 3, Getting Started with Web Applications. In the example application included with this tutorial, many of these steps have been completed for you and are listed here simply to show what needs to be done should you wish to create a similar application. The completed version of this example application can be found in the directory *tut-install*`/examples/security/hello2_basicauth/` .

1. Follow the steps in To Set Up Your System for Running the Security Examples.
2. Create a web module as described in Chapter 3, Getting Started with Web Applications for the servlet example, `hello2` .
3. Add the appropriate security annotations to the servlet. The security annotations are described in Specifying Security for Basic Authentication Using Annotations.
4. Build, package, and deploy the web application by following the steps in To Build, Package, and Deploy the Servlet Basic Authentication Example Using NetBeans IDE or To Build, Package, and Deploy the Servlet Basic Authentication Example Using Ant.
5. Run the web application by following the steps described in To Run the Basic Authentication Servlet.

Specifying Security for Basic Authentication Using Annotations

The default authentication mechanism used by the GlassFish Server is basic authentication. With basic authentication, the GlassFish Server spawns a standard login dialog to collect user name and password data for a protected resource. Once the user is authenticated, access to the protected resource is permitted.

To specify security for a servlet, use the `@ServletSecurity` annotation. This annotation allows you to specify both specific constraints on HTTP methods and more general constraints that apply to all HTTP methods for which no specific constraint is specified. Within the `@ServletSecurity` annotation, you can specify the following annotations:

- The `@HttpMethodConstraint` annotation, which applies to a specific HTTP method
- The more general `@HttpConstraint` annotation, which applies to all HTTP methods for which there is no corresponding `@HttpMethodConstraint` annotation

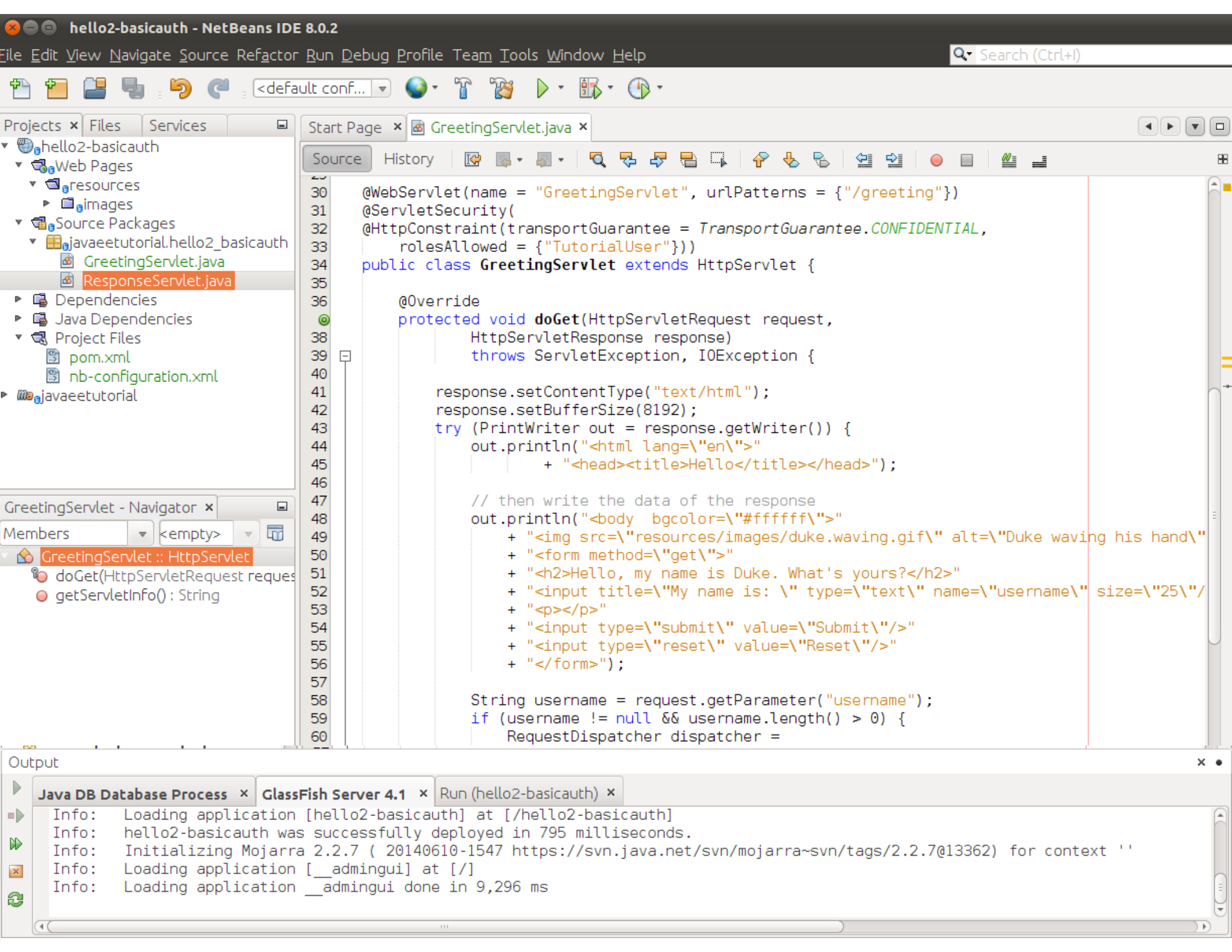
Both the `@HttpMethodConstraint` and `@HttpConstraint` annotations within the `@ServletSecurity` annotation can specify the following:

- A `transportGuarantee` element that specifies the data protection requirements (that is, whether or not SSL/TLS is required) that must be satisfied by the connections on which requests arrive. Valid values for this element are `NONE` and `CONFIDENTIAL` .
- A `rolesAllowed` element that specifies the names of the authorized roles.

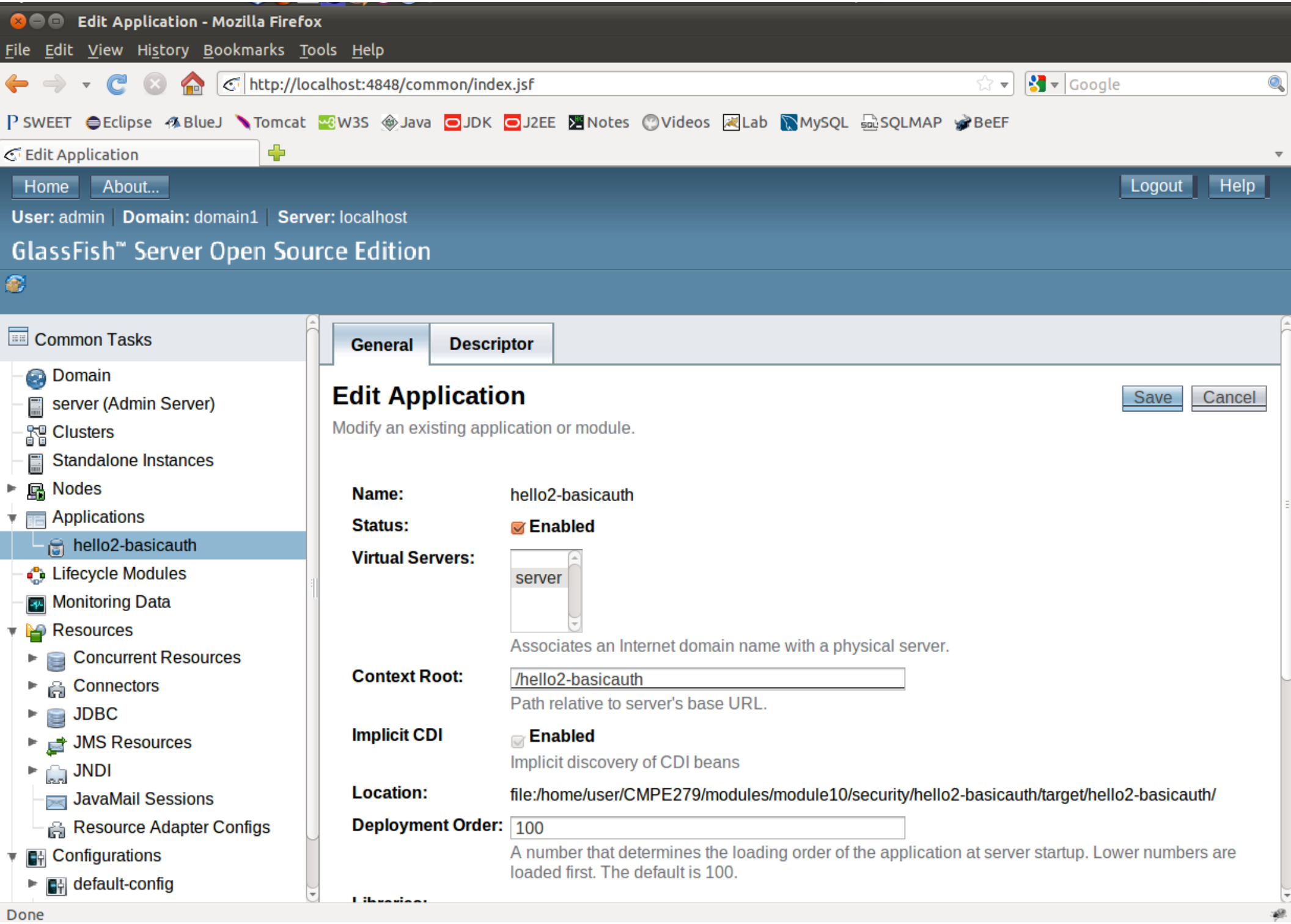
For the `hello2_basicauth` application, the `GreetingServlet` has the following annotations:

```
@WebServlet(name = "GreetingServlet", urlPatterns = {"/greeting"})
@ServletSecurity(
@HttpConstraint(transportGuarantee = TransportGuarantee.CONFIDENTIAL,
    rolesAllowed = {"TutorialUser"}))
```

These annotations specify that the request URI `/greeting` can be accessed only by users who have been authorized to access this URL because they have been verified to be in the role `TutorialUser` . The data will be sent over a protected transport in order to keep the user name and password data from being read in transit.


hello2-basicauth - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default conf...
Projects
Files
Services
hello2-basicauth
Web Pages
resources
images
Source Packages
javaeetutorial.hello2_basicauth
GreetingServlet.java
ResponseServlet.java
Dependencies
Java Dependencies
Project Files
pom.xml
nb-configuration.xml
javaeetutorial
GreetingServlet - Navigator
Members
<empty>
GreetingServlet :: HttpServlet
doGet(HttpServletRequest reques
getServletInfo() : String
Start Page
GreetingServlet.java
Source
History
@WebServlet(name = "GreetingServlet", urlPatterns = {"/greeting"})
@ServletSecurity(
@HttpConstraint(transportGuarantee = TransportGuarantee.CONFIDENTIAL,
rolesAllowed = {"TutorialUser"}))
public class GreetingServlet extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
response.setBufferSize(8192);
try (PrintWriter out = response.getWriter()) {
out.println("<html lang=\"en\">"
+ "<head><title>Hello</title></head>");
// then write the data of the response
out.println("<body bgcolor=\"#ffffff\">"
+ "<img src=\"resources/images/duke.waving.gif\" alt=\"Duke waving his hand\"
+ "<form method=\"get\">"
+ "<h2>Hello, my name is Duke. What's yours?</h2>"
+ "<input title=\"My name is: \" type=\"text\" name=\"username\" size=\"25\"/
+ " "
+ "<input type=\"submit\" value=\"Submit\"/>"
+ "<input type=\"reset\" value=\"Reset\"/>"
+ "</form>");
String username = request.getParameter("username");
if (username != null && username.length() > 0) {
RequestDispatcher dispatcher =
Output
Java DB Database Process
GlassFish Server 4.1
Run (hello2-basicauth)
Info: Loading application [hello2-basicauth] at [/hello2-basicauth]
Info: hello2-basicauth was successfully deployed in 795 milliseconds.
Info: Initializing Mojarra 2.2.7 (20140610-1547 https://svn.java.net/svn/mojarra~svn/tags/2.2.7@13362) for context ''
Info: Loading application [__admingui] at [/]
Info: Loading application __admingui done in 9,296 ms

Login: admin / 123456


Edit Application - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:4848/common/index.jsf
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF
Edit Application
Home About... Logout Help
User: admin | Domain: domain1 | Server: localhost
GlassFish™ Server Open Source Edition
Common Tasks
Domain
server (Admin Server)
Clusters
Standalone Instances
Nodes
Applications
hello2-basicauth
Lifecycle Modules
Monitoring Data
Resources
Concurrent Resources
Connectors
JDBC
JMS Resources
JNDI
JavaMail Sessions
Resource Adapter Configs
Configurations
default-config
General Descriptor
Edit Application
Modify an existing application or module.
Save Cancel
Name: hello2-basicauth
Status: Enabled
Virtual Servers: server
Associates an Internet domain name with a physical server.
Context Root: /hello2-basicauth
Path relative to server's base URL.
Implicit CDI Enabled
Implicit discovery of CDI beans
Location: file:/home/user/CMPE279/modules/module10/security/hello2-basicauth/target/hello2-basicauth/
Deployment Order: 100
A number that determines the loading order of the application at server startup. Lower numbers are loaded first. The default is 100.
Done

To Run the Basic Authentication Servlet

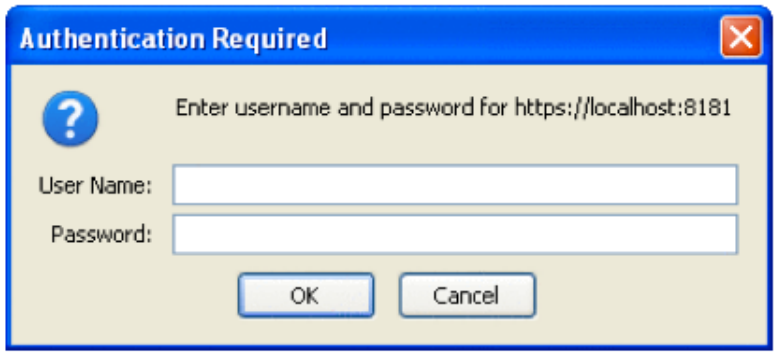
1. In a web browser, navigate to the following URL:

   ```
   https://localhost:8181/hello2_basicauth/greeting
   ```

   You may be prompted to accept the security certificate for the server. If so, accept the security certificate. If the browser warns that the certificate is invalid because it is self-signed, add a security exception for the application.

   An Authentication Required dialog box appears. Its appearance varies, depending on the browser you use. Figure 25–6 shows an example.

   Figure 25–6 Sample Basic Authentication Dialog Box

2. Type a user name and password combination that corresponds to a user who has already been created in the `file` realm of the GlassFish Server and has been assigned to the group of `TutorialUser` ; then click OK.

   Basic authentication is case sensitive for both the user name and password, so type the user name and password exactly as defined for the GlassFish Server.

   The server returns the requested resource if all the following conditions are met.

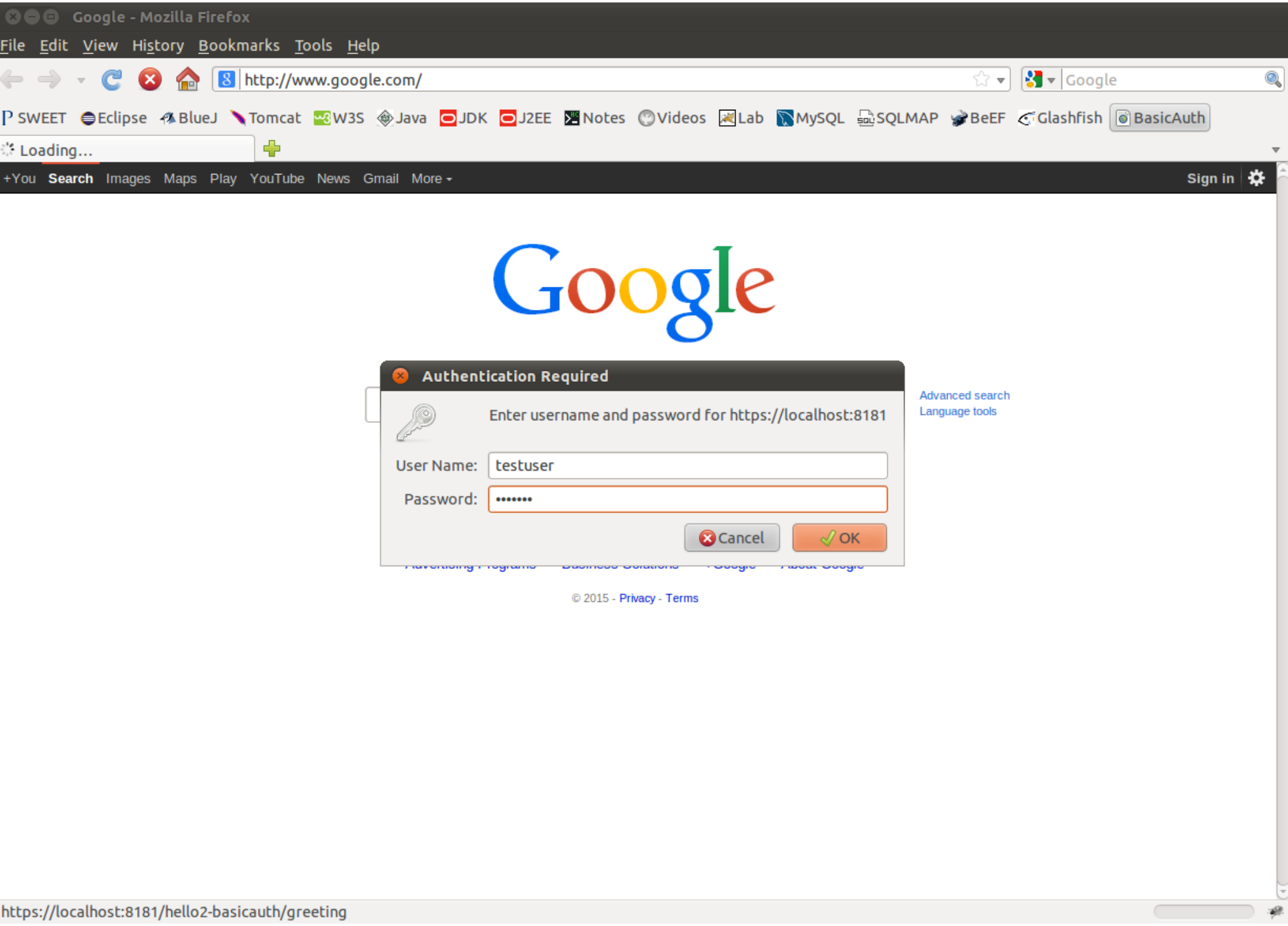
   - A user with the user name you entered is defined for the GlassFish Server.
   - The user with the user name you entered has the password you entered.
   - The user name and password combination you entered is assigned to the group `TutorialUser` on the GlassFish Server.
   - The role of `TutorialUser` , as defined for the application, is mapped to the group `TutorialUser` , as defined for the GlassFish Server.

   When these conditions are met and the server has authenticated the user, the application will appear as shown in Figure 3–2 but with a different URL.

3. Type a name in the text field and click the Submit button.

   Because you have already been authorized, the name you enter in this step does not have any limitations. You have unlimited access to the application now.

   The application responds by saying "Hello" to you, as shown in Figure 3–3 but with a different URL.


Google - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.google.com/
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth
Loading...
+You Search Images Maps Play YouTube News Gmail More
Sign in
Google
Authentication Required
Enter username and password for https://localhost:8181
User Name: testuser
Password: •••••••
Cancel
OK
Advanced search
Language tools
© 2015 - Privacy - Terms
https://localhost:8181/hello2-basicauth/greeting

Hello - Mozilla Firefox

File Edit View History Bookmarks Tools Help

localhost https://localhost:8181/hello2-basicauth/greeting

Google

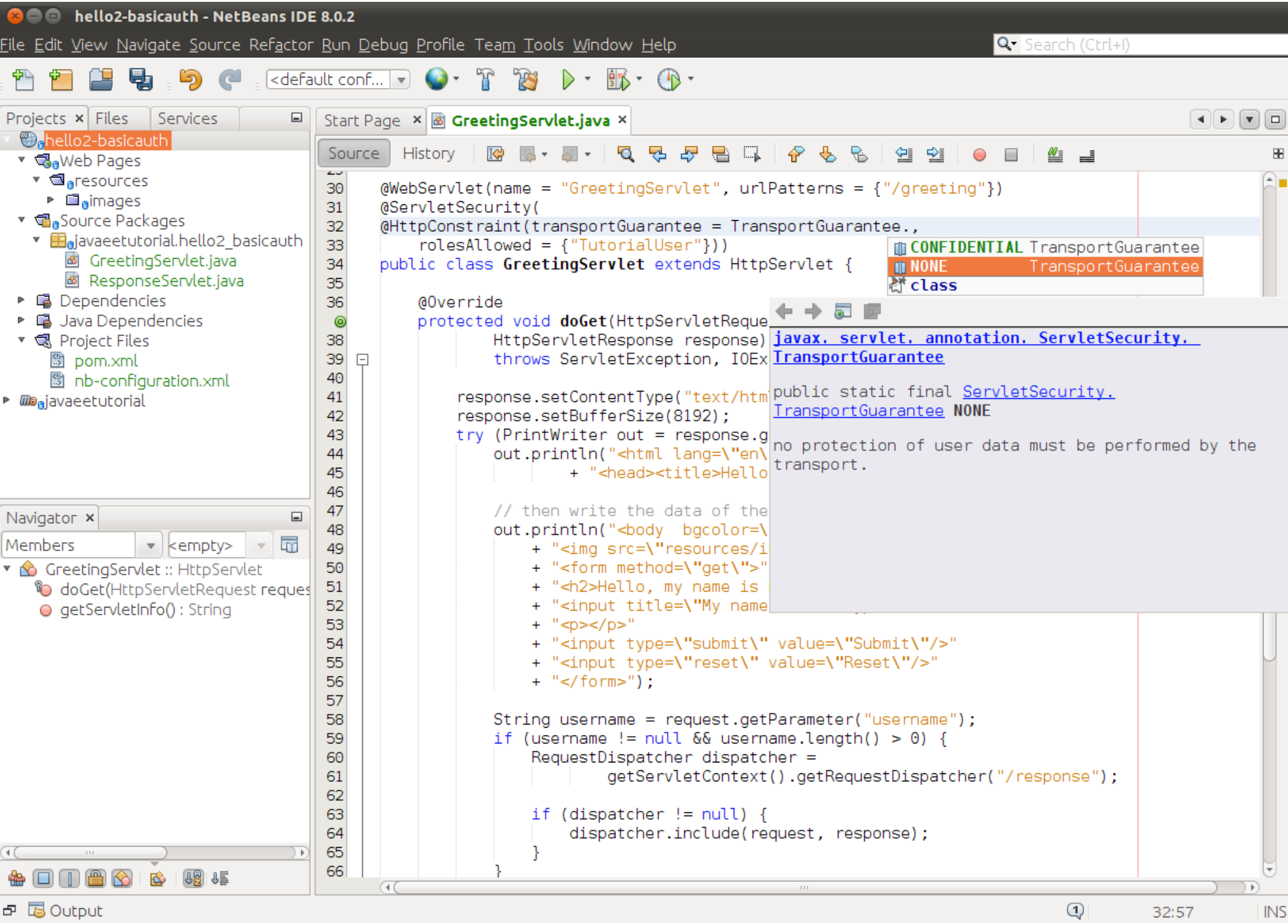
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth

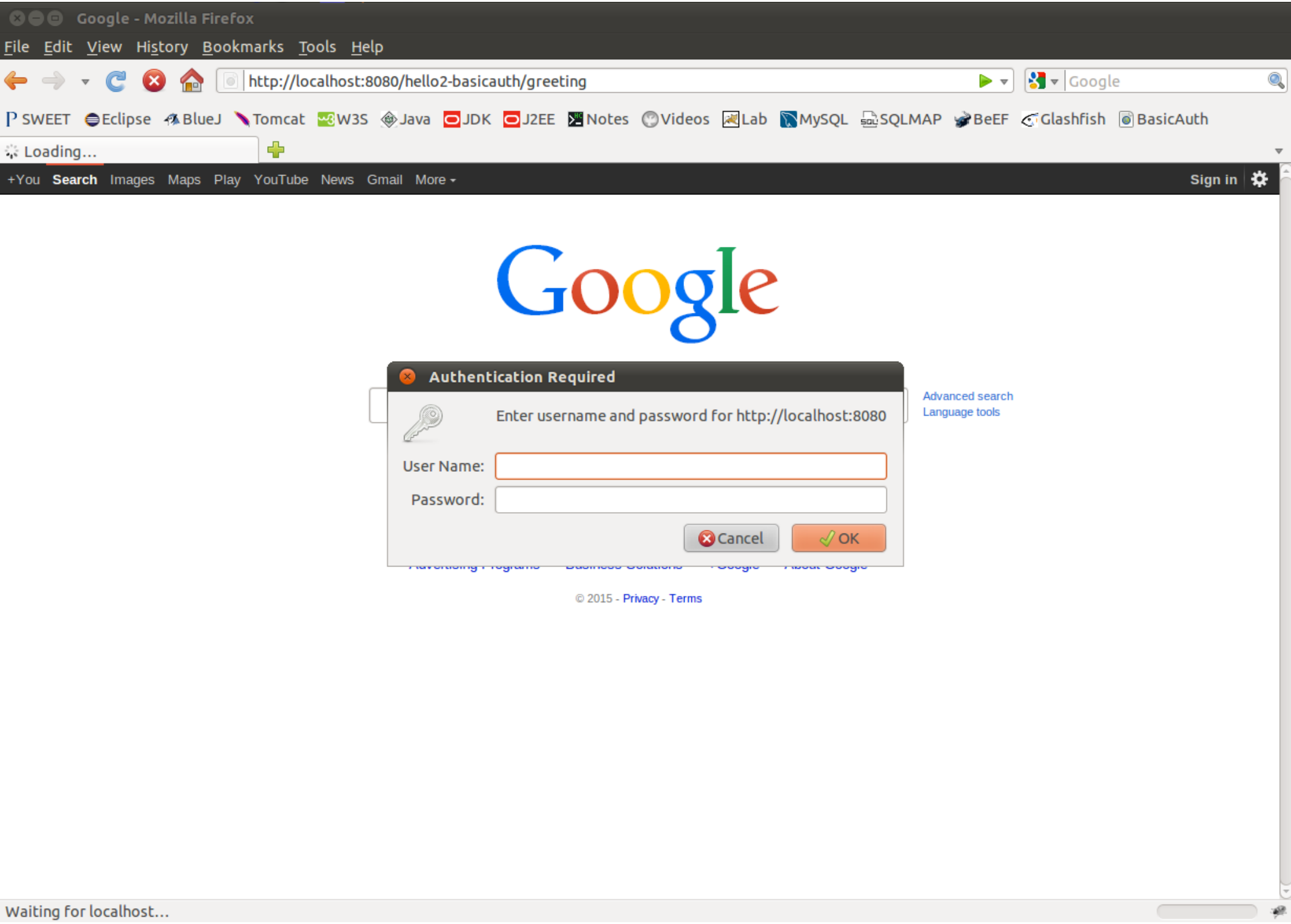
Hello

Hello, my name is Duke. What's yours?

Submit Reset

Done


hello2-basicauth - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Projects
Files
Services
hello2-basicauth
Web Pages
resources
images
Source Packages
javaeetutorial.hello2_basicauth
GreetingServlet.java
ResponseServlet.java
Dependencies
Java Dependencies
Project Files
pom.xml
nb-configuration.xml
javaeetutorial
Navigator
Members
<empty>
GreetingServlet :: HttpServlet
doGet(HttpServletRequest reques
getServletInfo() : String
Start Page
GreetingServlet.java
Source
History
@WebServlet(name = "GreetingServlet", urlPatterns = {"/greeting"})
@ServletSecurity(
@HttpConstraint(transportGuarantee = TransportGuarantee.,
rolesAllowed = {"TutorialUser"}))
public class GreetingServlet extends HttpServlet {
@Override
protected void doGet(HttpServletReque
HttpServletResponse response)
throws ServletException, IOEx
response.setContentType("text/htm
response.setBufferSize(8192);
try (PrintWriter out = response.g
out.println("<html lang=\"en\
+ "<head><title>Hello
// then write the data of the
out.println("<body bgcolor=\
+ "<img src=\"resources/i
+ "<form method=\"get\">"
+ "<h2>Hello, my name is
+ "<input title=\"My name
+ " "
+ "<input type=\"submit\" value=\"Submit\"/>"
+ "<input type=\"reset\" value=\"Reset\"/>"
+ "</form>");
String username = request.getParameter("username");
if (username != null && username.length() > 0) {
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/response");
if (dispatcher != null) {
dispatcher.include(request, response);
}
}
CONFIDENTIAL TransportGuarantee
NONE TransportGuarantee
class
javax.servlet.annotation.ServletSecurity.TransportGuarantee
public static final ServletSecurity.TransportGuarantee NONE
no protection of user data must be performed by the transport.
Output
32:57
INS


Google - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello2-basicauth/greeting
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth
Loading...
+You Search Images Maps Play YouTube News Gmail More
Sign in
Google
Authentication Required
Enter username and password for http://localhost:8080
User Name:
Password:
Cancel
OK
Advanced search
Language tools
© 2015 - Privacy - Terms
Waiting for localhost...

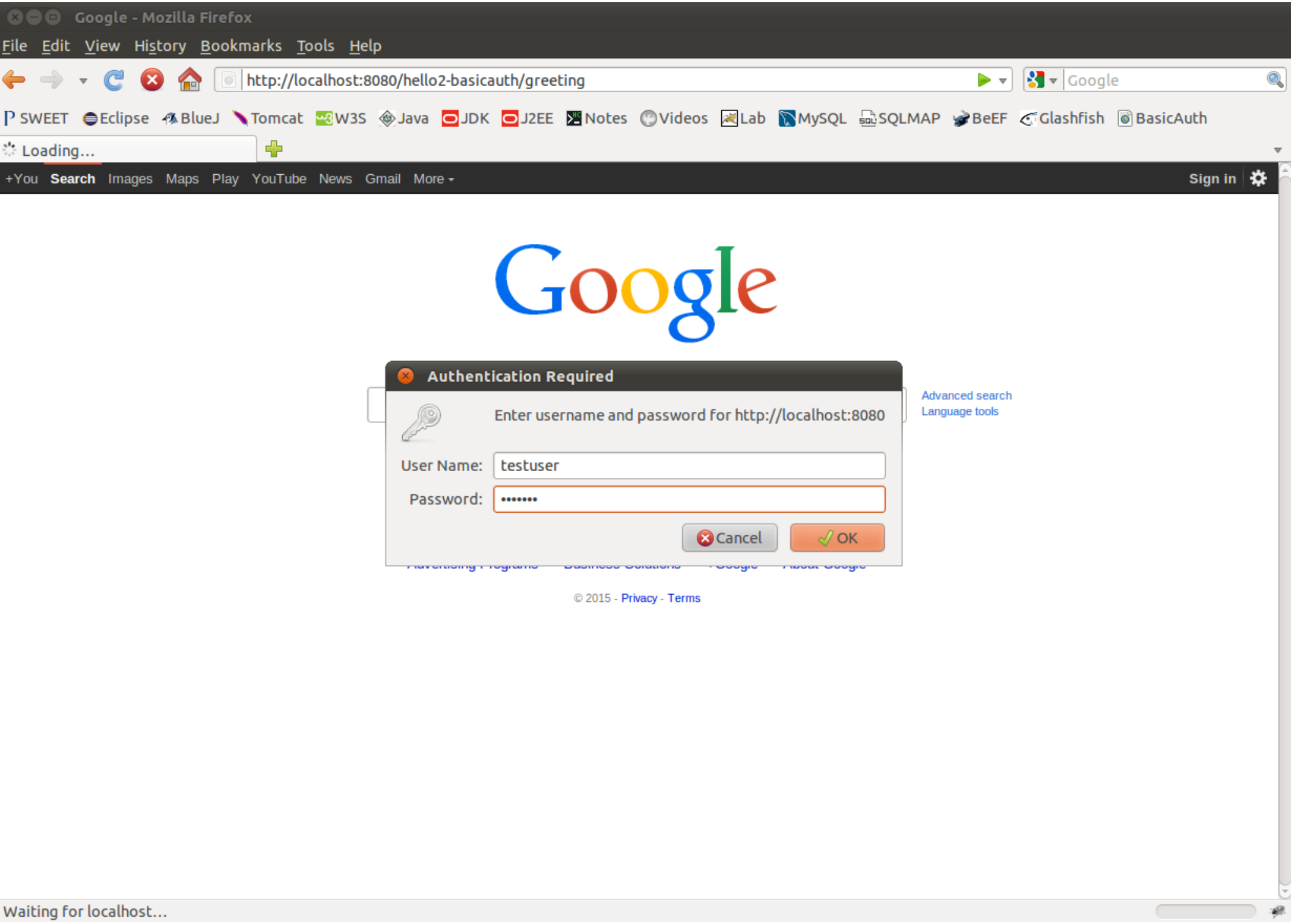
Request | Response | Trap

```
GET http://localhost:8080/hello2-basicauth/greeting HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.12) Gecko/20101027 Ubuntu/10.10 (maverick) Firefox/3.6.12 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
Cookie: BEEFHOOK=9dATRZww6XVaZ8XZHqjI0sgoMqNBx4INhRaB0uR7DEmWZBnyDnTZSYI3O5gPaN8qv0L6GRZVLgR3y9hl
```

Request | Response | Trap

```
HTTP/1.1 401 Unauthorized
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/Oracle Corporation/1.7)
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 19:00:00 EST
WWW-Authenticate: Basic realm=""
Content-Language:
Content-Type: text/html
Date: Wed, 11 Mar 2015 04:51:18 GMT
Content-Length: 1090
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html xmlns="http://www.w3.org/1999/xhtml"><head><title>GlassFish Server Open Source Edition  4.1  - Error report</title><style type="text/css"><!--H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}A {color : black;}HR {color : #525D76;}--></style> </head><body><h1>HTTP Status 401 - Unauthorized</h1><hr/><p><b>type</b> Status report</p><p><b>message</b>Unauthorized</p><p><b>description</b>This request requires HTTP authentication.</p><hr/><h3>GlassFish Server Open Source Edition  4.1 </h3></body></html>
```


Google - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello2-basicauth/greeting
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth
Loading...
+You Search Images Maps Play YouTube News Gmail More
Sign in
Google
Authentication Required
Enter username and password for http://localhost:8080
User Name: testuser
Password: •••••••
Cancel
OK
Advanced search
Language tools
© 2015 - Privacy - Terms
Waiting for localhost...

Hello - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/hello2-basicauth/greeting

Google

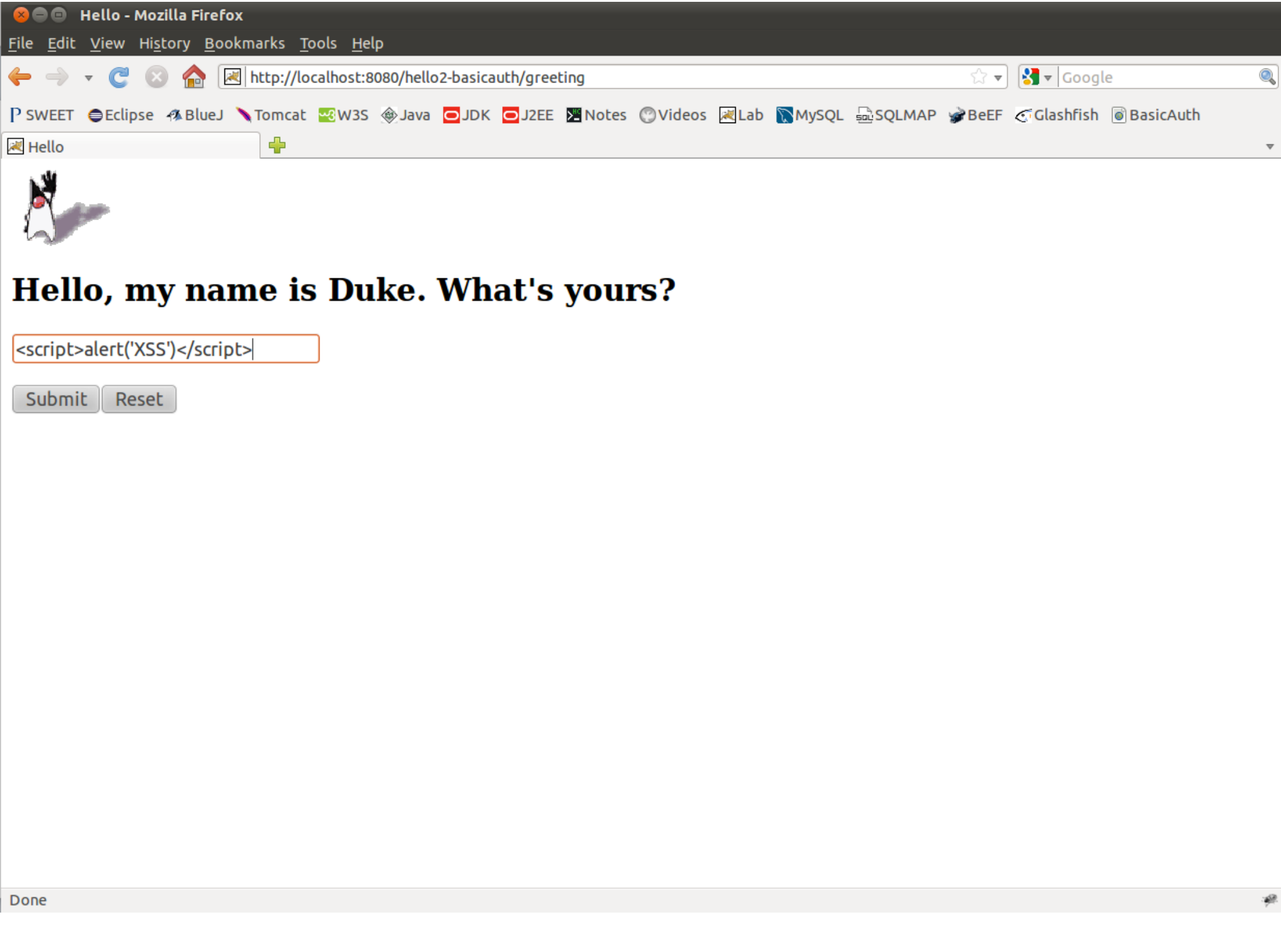
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth

Hello

Hello, my name is Duke. What's yours?

Submit Reset

Done


Hello - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello2-basicauth/greeting
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth
Hello
Hello, my name is Duke. What's yours?
<script>alert('XSS')</script>
Submit Reset
Done

Hello - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/hello2-basicauth/greeting?username=<script>alert('XSS')<%2Fscript>

Google

SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish BasicAuth

Hello

Hello, my name is Duke. What's yours?

Submit Reset

Hello,

The page at http://localhost:8080 says:

XSS

OK

Transferring data from localhost...

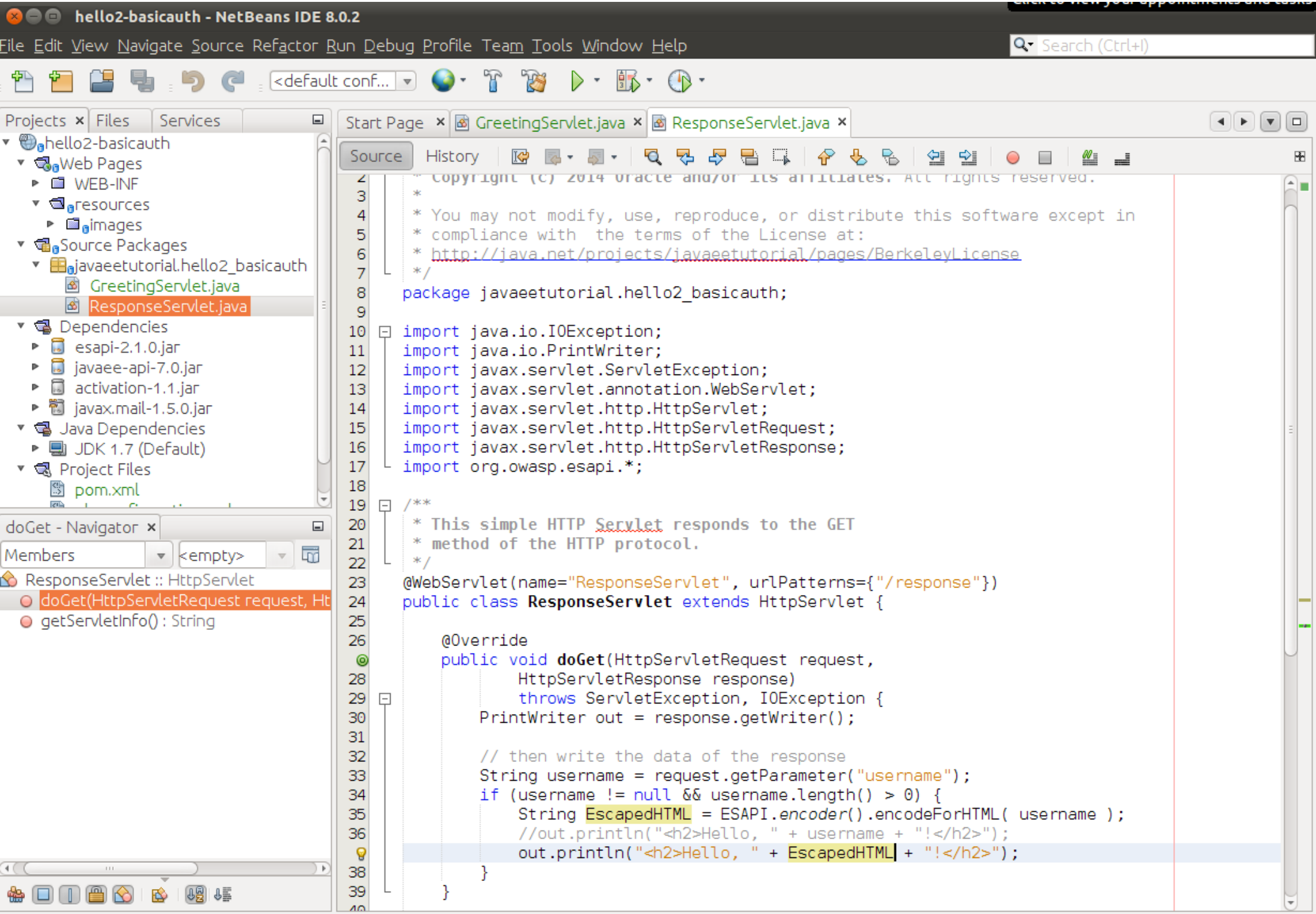
Request | Response | Trap

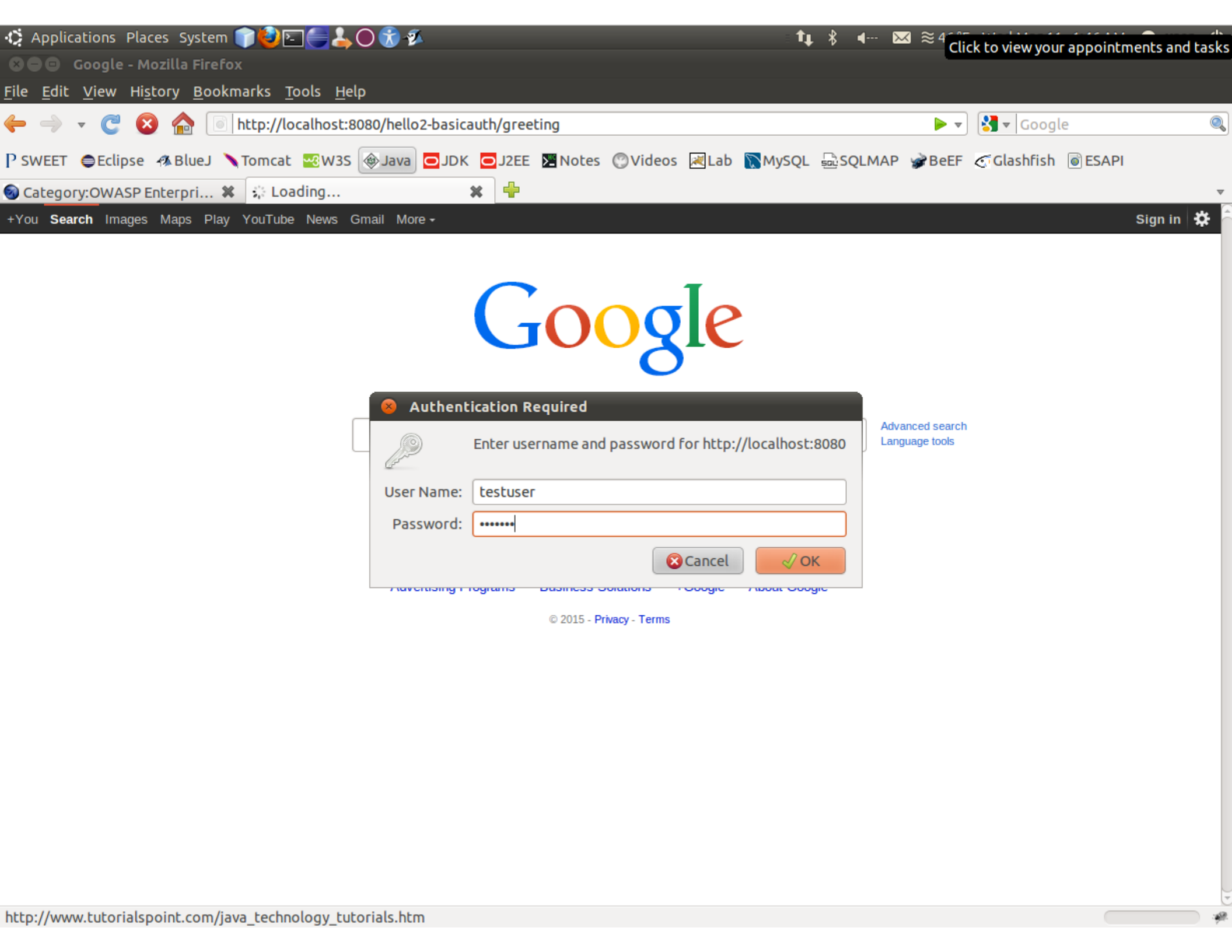
```
GET http://localhost:8080/hello2-basicauth/greeting HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.12) Gecko/20101027 Ubuntu/10.10 (maverick) Firefox/3.6.12 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
Cookie: BEEFHOOK=9dATRZww6XVaZ8XZHqjI0sgoMqNBx4INhRaBOuR7DEmWZBnyDnTZSYI3O5gPaN8qv0L6GRZVLgR3y9hl
Authorization: Basic dGVzdHVzZXI6d2VsY29tZQ==
```

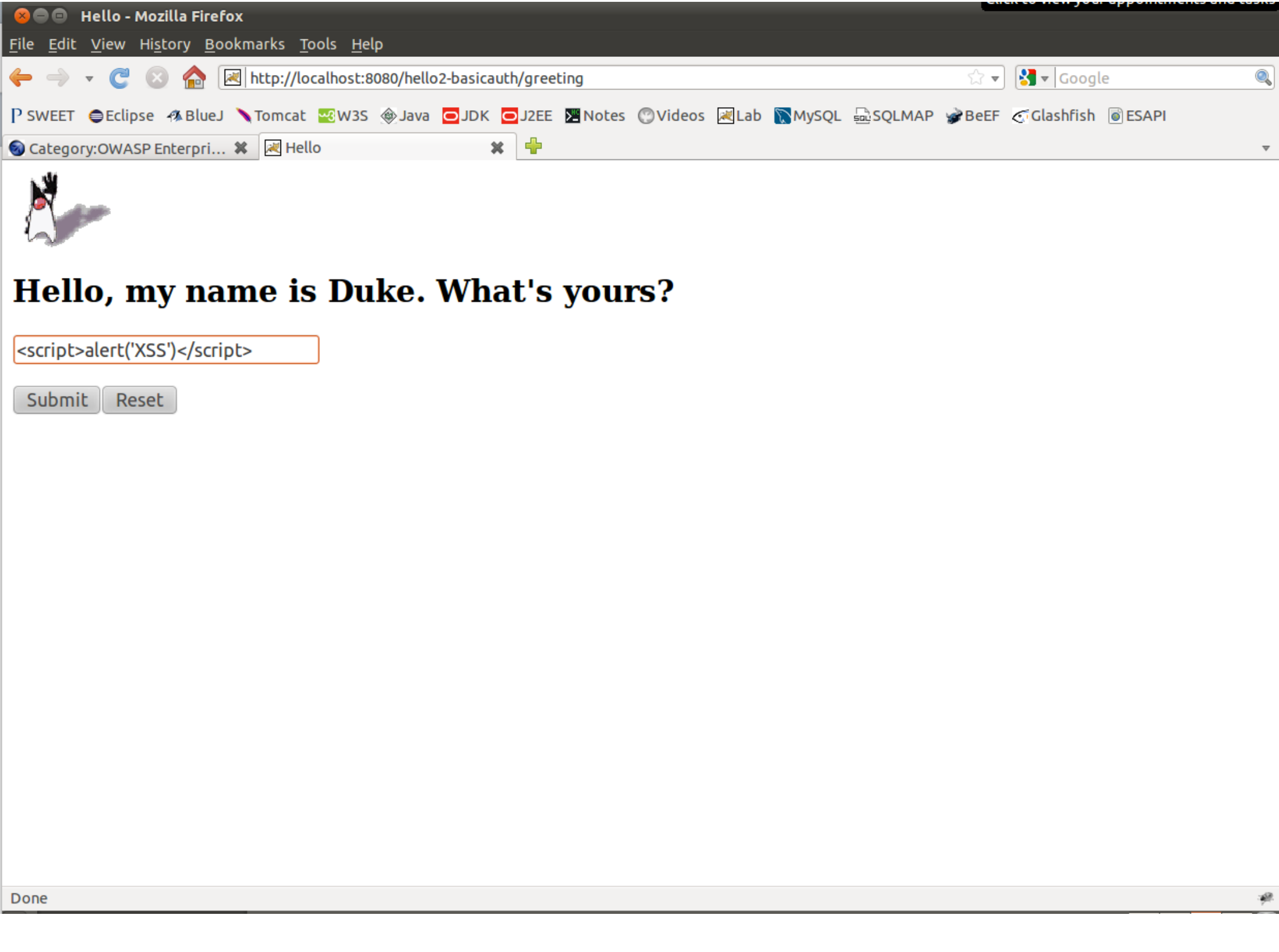
Request | Response | Trap

```
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/Oracle Corporation/1.7)
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 19:00:00 EST
Content-Type: text/html;charset=ISO-8859-1
Date: Wed, 11 Mar 2015 04:47:41 GMT
Content-Length: 427
```

```
<html lang="en"><head><title>Hello</title></head>
<body  bgcolor="#ffffff"><img src="resources/images/duke.waving.gif" alt="Duke waving his hand"><form method="get"><h2>Hello, my name is
Duke. What's yours?</h2><input title="My name is: " type="text" name="username" size="25"/><p></p><input type="submit" value="Submit"/><
input type="reset" value="Reset"/></form>
<h2>Hello, <script>alert('XSS')</script>!</h2>
</body></html>
```


hello2-basicauth - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default conf...
Projects
Files
Services
hello2-basicauth
Web Pages
WEB-INF
resources
images
Source Packages
javaeetutorial.hello2_basicauth
GreetingServlet.java
ResponseServlet.java
Dependencies
esapi-2.1.0.jar
javaee-api-7.0.jar
activation-1.1.jar
javax.mail-1.5.0.jar
Java Dependencies
JDK 1.7 (Default)
Project Files
pom.xml
doGet - Navigator
Members
<empty>
ResponseServlet :: HttpServlet
doGet(HttpServletRequest request, Ht
getServletInfo() : String
Start Page
GreetingServlet.java
ResponseServlet.java
Source
History
* Copyright (c) 2014 Oracle and/or its affiliates. All rights reserved.
*
* You may not modify, use, reproduce, or distribute this software except in
* compliance with the terms of the License at:
* http://java.net/projects/javaeetutorial/pages/BerkeleyLicense
*/
package javaeetutorial.hello2_basicauth;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.owasp.esapi.*;
/**
* This simple HTTP Servlet responds to the GET
* method of the HTTP protocol.
*/
@WebServlet(name="ResponseServlet", urlPatterns={"/response"})
public class ResponseServlet extends HttpServlet {
@Override
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
PrintWriter out = response.getWriter();
// then write the data of the response
String username = request.getParameter("username");
if (username != null && username.length() > 0) {
String EscapedHTML = ESAPI.encoder().encodeForHTML(username);
//out.println("<h2>Hello, " + username + "!</h2>");
out.println("<h2>Hello, " + EscapedHTML + "!</h2>");
}
}


Applications Places System
Click to view your appointments and tasks
Google - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello2-basicauth/greeting
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Category:OWASP Enterpri...
Loading...
+You Search Images Maps Play YouTube News Gmail More
Sign in
Google
Authentication Required
Enter username and password for http://localhost:8080
User Name: testuser
Password: ••••••••
Cancel
OK
Advanced search
Language tools
© 2015 - Privacy - Terms
http://www.tutorialspoint.com/java_technology_tutorials.htm


Hello - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello2-basicauth/greeting
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Category:OWASP Enterpri...
Hello
Hello, my name is Duke. What's yours?
<script>alert('XSS')</script>
Submit
Reset
Done

Hello - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/hello2-basicauth/greeting?username=<script>alert('XSS')<%2Fscript>

Google

SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI

Hello

Hello, my name is Duke. What's yours?

Submit Reset

Done

Request | Response | Trap

```
GET http://localhost:8080/hello2-basicauth/greeting?username=%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.12) Gecko/20101027 Ubuntu/10.10 (maverick) Firefox/3.6.12 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
Referer: http://localhost:8080/hello2-basicauth/greeting
Cookie: BEEFHOOK=9dATRZww6XVaZ8XZHqjI0sgoMqNBx4INhRaB0uR7DEmWZBnyDnTZSYI3O5gPaN8qv0L6GRZVLgR3y9hl
Authorization: Basic dGVzdHVzZXI6d2VsY29tZQ==
```

Request | Response | Trap

```
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/Oracle Corporation/1.7)
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 19:00:00 EST
Content-Type: text/html;charset=ISO-8859-1
Date: Wed, 11 Mar 2015 05:49:06 GMT
Content-Length: 365
```

```
<html lang="en"><head><title>Hello</title></head>
<body  bgcolor="#ffffff"><img src="resources/images/duke.waving.gif" alt="Duke waving his hand"><form method="get"><h2>Hello, my name is
Duke. What's yours?</h2><input title="My name is: " type="text" name="username" size="25"/><p></p><input type="submit" value="Submit"/><
input type="reset" value="Reset"/></form>
```

ORACLE

The Java EE 6 Tutorial

« **Previous**: Example: Basic Authentication with a Servlet **Next**: Chapter 26 Getting Started Securing Enterprise Applications »

Example: Form-Based Authentication with a JavaServer Faces Application

This example explains how to use form-based authentication with a JavaServer Faces application. With form-based authentication, you can customize the login screen and error pages that are presented to the web client for authentication of the user name and password. When a user submits his or her name and password, the server determines whether the user name and password are those of an authorized user and, if authorized, sends the requested web resource.

This example, `hello1_formauth`, adds security to the basic JavaServer Faces application shown in Web Modules: The `hello1` Example.

In general, the steps necessary for adding form-based authentication to an unsecured JavaServer Faces application are similar to those described in Example: Basic Authentication with a Servlet. The major difference is that you must use a deployment descriptor to specify the use of form-based authentication, as described in Specifying Security for the Form-Based Authentication Example. In addition, you must create a login form page and a login error page, as described in Creating the Login Form and the Error Page.

The completed version of this example application can be found in the directory *tut-install*`/examples/security/hello1_formauth/`.

Creating the Login Form and the Error Page

When using form-based login mechanisms, you must specify a page that contains the form you want to use to obtain the user name and password, as well as a page to display if login authentication fails. This section discusses the login form and the error page used in this example. Specifying Security for the Form-Based Authentication Example shows how you specify these pages in the deployment descriptor.

The login page can be an HTML page, a JavaServer Faces or JSP page, or a servlet, and it must return an HTML page containing a form that conforms to specific naming conventions (see the Java Servlet 3.0 specification for more information on these requirements). To do this, include the elements that accept user name and password information between `<form></form>` tags in your login page. The content of an HTML page, JavaServer Faces or JSP page, or servlet for a login page should be coded as follows:

```
<form method=post action="j_security_check">
    <input type="text" name="j_username">
    <input type="password" name= "j_password">
</form>
```

The full code for the login page used in this example can be found at *tut-install*`/examples/security/hello1_formauth/web/login.xhtml` . An example of the running login form page is shown later, in Figure 25–7. Here is the code for this page:

```
<html xmlns="http://www.w3.org/1999/xhtml"
```

The login error page is displayed if the user enters a user name and password combination that is not authorized to access the protected URI. For this example, the login error page can be found at *tut-install*`/examples/security/hello1_formauth/web/error.xhtml` . For this example, the login error page explains the reason for receiving the error page and provides a link that will allow the user to try again. Here is the code for this page:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Login Error</title>
    </h:head>
    <h:body>
    <h2>Invalid user name or password.</h2>

    <p>Please enter a user name or password that is authorized to access this
        application. For this application, this means a user that has been
        created in the <code>file</code> realm and has been assigned to the
        <em>group</em> of <code>TutorialUser</code>.</p>
   <h:link outcome="login">Return to login page</h:link>

    </h:body>
</html>
```

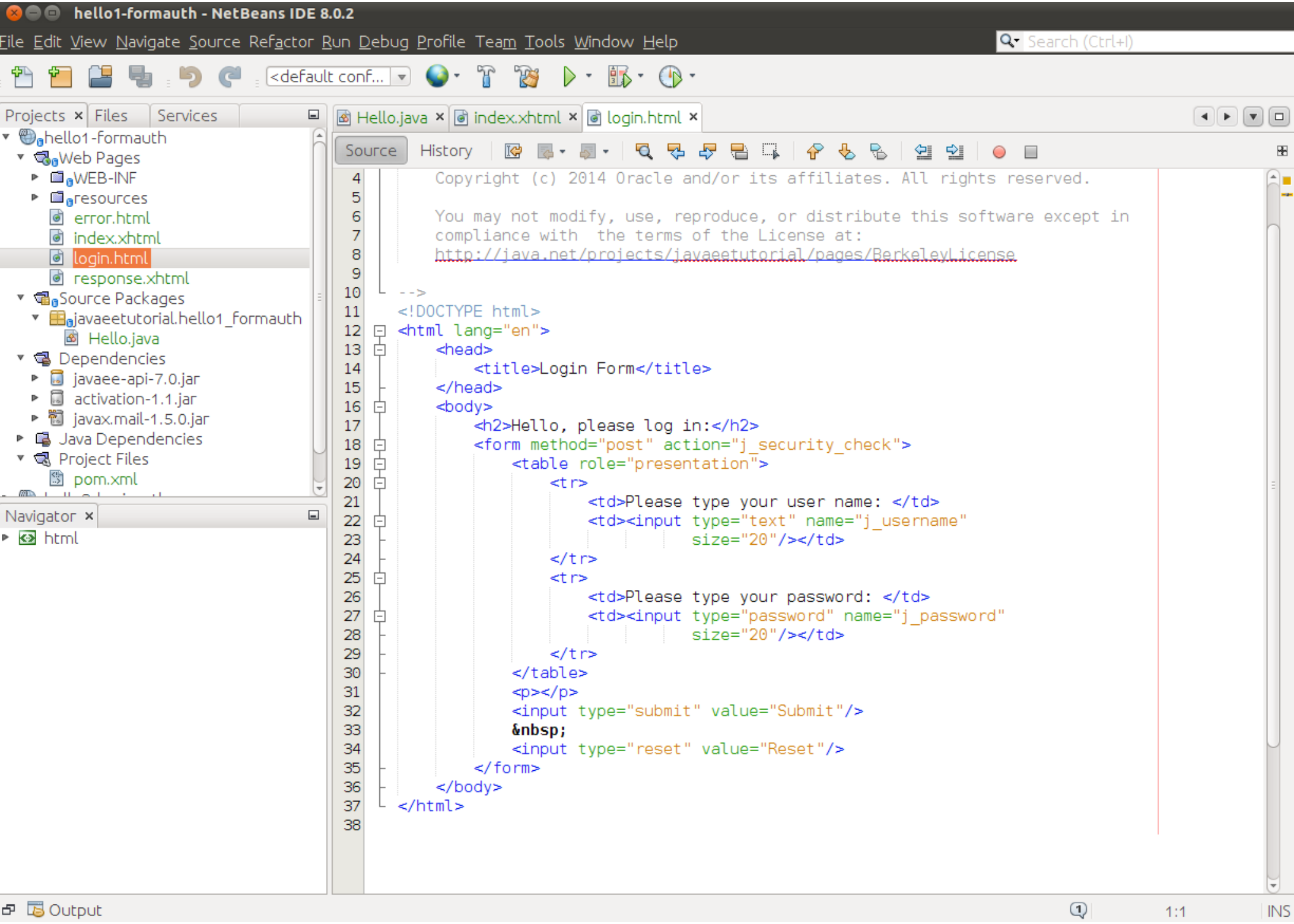
</ht

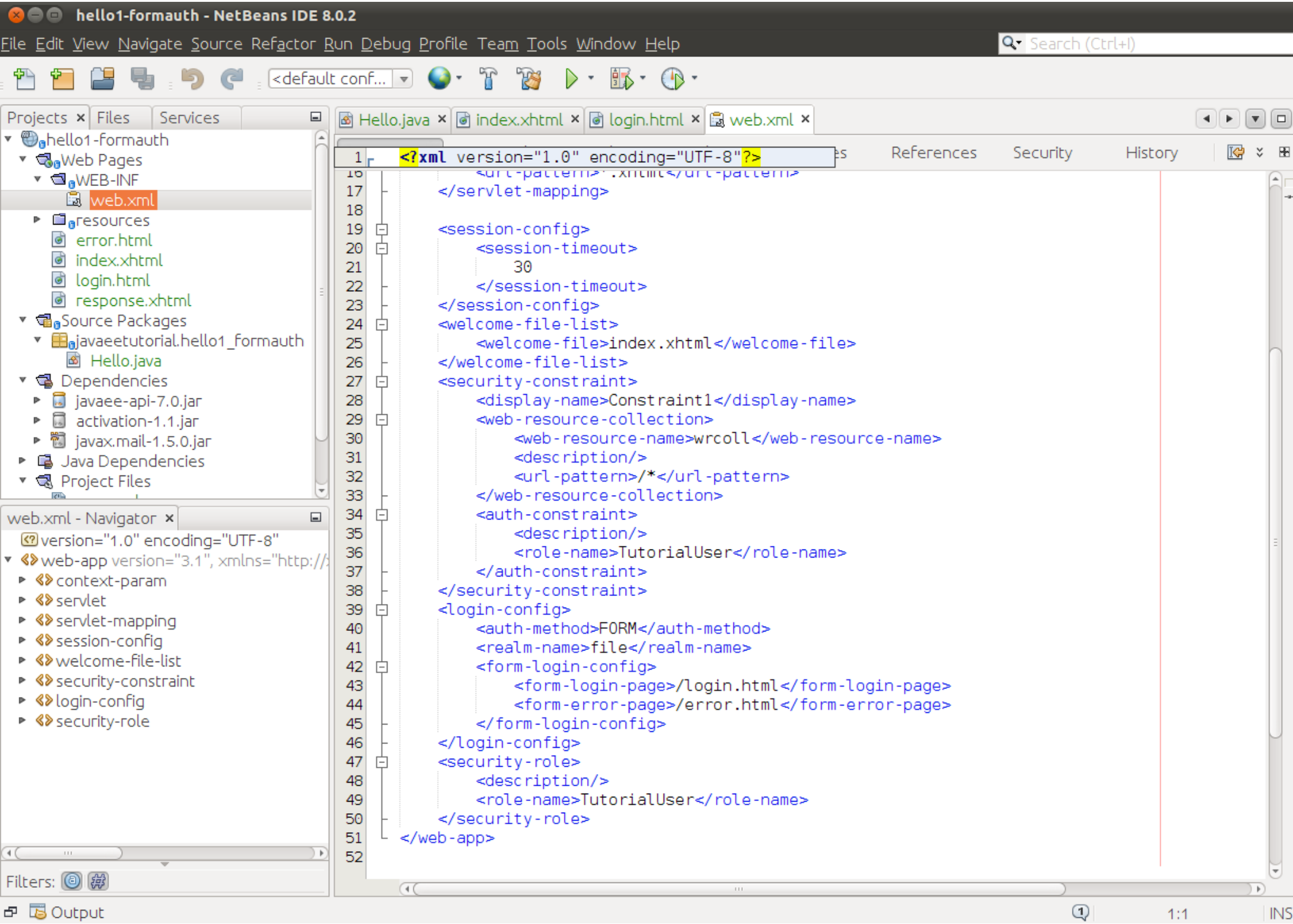
Specifying Security for the Form-Based Authentication Example

This example takes a very simple servlet-based web application and adds form-based security. To specify form-based instead of basic authentication for a JavaServer Faces example, you must use the deployment descriptor.

The following sample code shows the security elements added to the deployment descriptor for this example, which can be found in *tut-install*/examples/security/hello1_formauth/web/WEB-INF/web.xml .

```
<security-constraint>
    <display-name>Constraint1</display-name>
    <web-resource-collection>
        <web-resource-name>wrcoll</web-resource-name>
        <description/>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>TutorialUser</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>file</realm-name>
    <form-login-config>
        <form-login-page>/login.xhtml</form-login-page>
        <form-error-page>/error.xhtml</form-error-page>
    </form-login-config>
</login-config>

<security-role>
    <description/>
    <role-name>TutorialUser</role-name>
</security-role>
```


hello1-formauth - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default conf...
Projects
Files
Services
hello1-formauth
Web Pages
WEB-INF
resources
error.html
index.xhtml
login.html
response.xhtml
Source Packages
javaeetutorial.hello1_formauth
Hello.java
Dependencies
javaee-api-7.0.jar
activation-1.1.jar
javax.mail-1.5.0.jar
Java Dependencies
Project Files
pom.xml
Navigator
html
Hello.java
index.xhtml
login.html
Source
History
4 Copyright (c) 2014 Oracle and/or its affiliates. All rights reserved.
5
6 You may not modify, use, reproduce, or distribute this software except in
7 compliance with the terms of the License at:
8 http://java.net/projects/javaeetutorial/pages/BerkeleyLicense
9
10 -->
11 <!DOCTYPE html>
12 <html lang="en">
13 <head>
14 <title>Login Form</title>
15 </head>
16 <body>
17 <h2>Hello, please log in:</h2>
18 <form method="post" action="j_security_check">
19 <table role="presentation">
20 <tr>
21 <td>Please type your user name: </td>
22 <td><input type="text" name="j_username"
23 size="20"/></td>
24 </tr>
25 <tr>
26 <td>Please type your password: </td>
27 <td><input type="password" name="j_password"
28 size="20"/></td>
29 </tr>
30 </table>
31
32 <input type="submit" value="Submit"/>
33
34 <input type="reset" value="Reset"/>
35 </form>
36 </body>
37 </html>
38
Output
1:1
INS


hello1-formauth - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default conf...
Projects
Files
Services
hello1-formauth
Web Pages
WEB-INF
web.xml
resources
error.html
index.xhtml
login.html
response.xhtml
Source Packages
javaeetutorial.hello1_formauth
Hello.java
Dependencies
javaee-api-7.0.jar
activation-1.1.jar
javax.mail-1.5.0.jar
Java Dependencies
Project Files
web.xml - Navigator
version="1.0" encoding="UTF-8"
web-app version="3.1", xmlns="http://
context-param
servlet
servlet-mapping
session-config
welcome-file-list
security-constraint
login-config
security-role
Filters:
Hello.java
index.xhtml
login.html
web.xml
References
Security
History
1 <?xml version="1.0" encoding="UTF-8"?>
17 </servlet-mapping>
18
19 <session-config>
20 <session-timeout>
21 30
22 </session-timeout>
23 </session-config>
24 <welcome-file-list>
25 <welcome-file>index.xhtml</welcome-file>
26 </welcome-file-list>
27 <security-constraint>
28 <display-name>Constraint1</display-name>
29 <web-resource-collection>
30 <web-resource-name>wrcoll</web-resource-name>
31 <description/>
32 <url-pattern>/*</url-pattern>
33 </web-resource-collection>
34 <auth-constraint>
35 <description/>
36 <role-name>TutorialUser</role-name>
37 </auth-constraint>
38 </security-constraint>
39 <login-config>
40 <auth-method>FORM</auth-method>
41 <realm-name>file</realm-name>
42 <form-login-config>
43 <form-login-page>/login.html</form-login-page>
44 <form-error-page>/error.html</form-error-page>
45 </form-login-config>
46 </login-config>
47 <security-role>
48 <description/>
49 <role-name>TutorialUser</role-name>
50 </security-role>
51 </web-app>
52
Output
1:1
INS

To Run the Form-Based Authentication Example

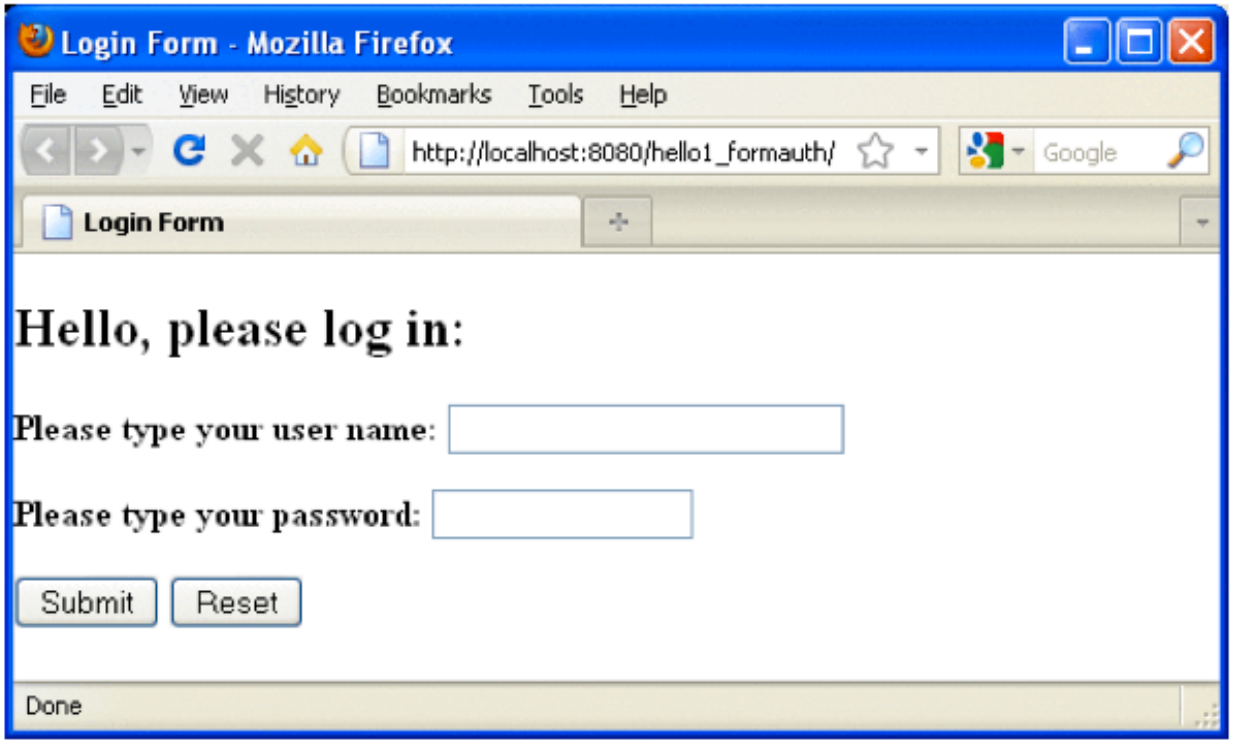
▼To run the web client for `hello1_formauth` , follow these steps.

1. Open a web browser to the following URL:

   `https://localhost:8181/hello1_formauth/`

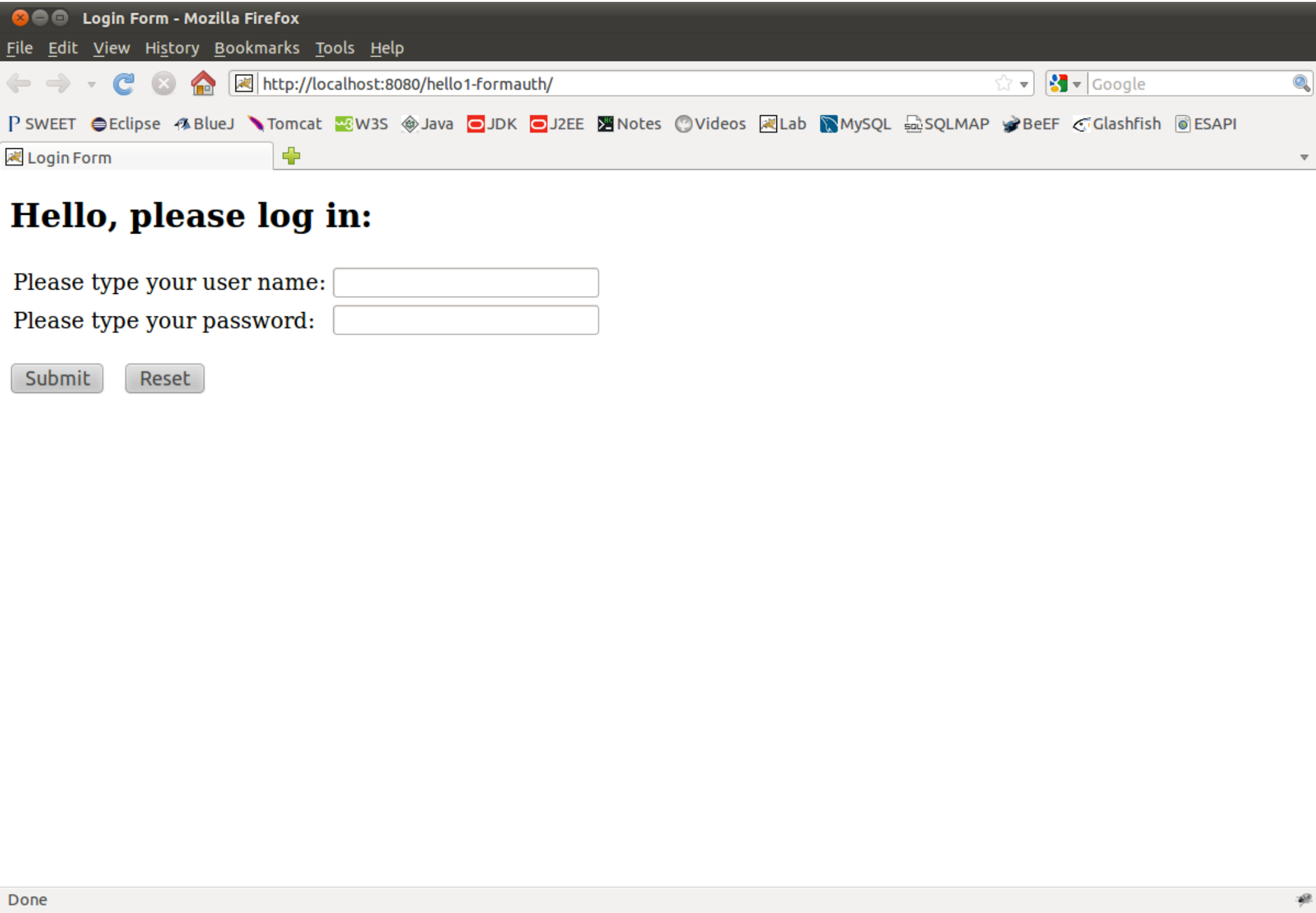
   The login form displays in the browser, as shown in Figure 25–7.

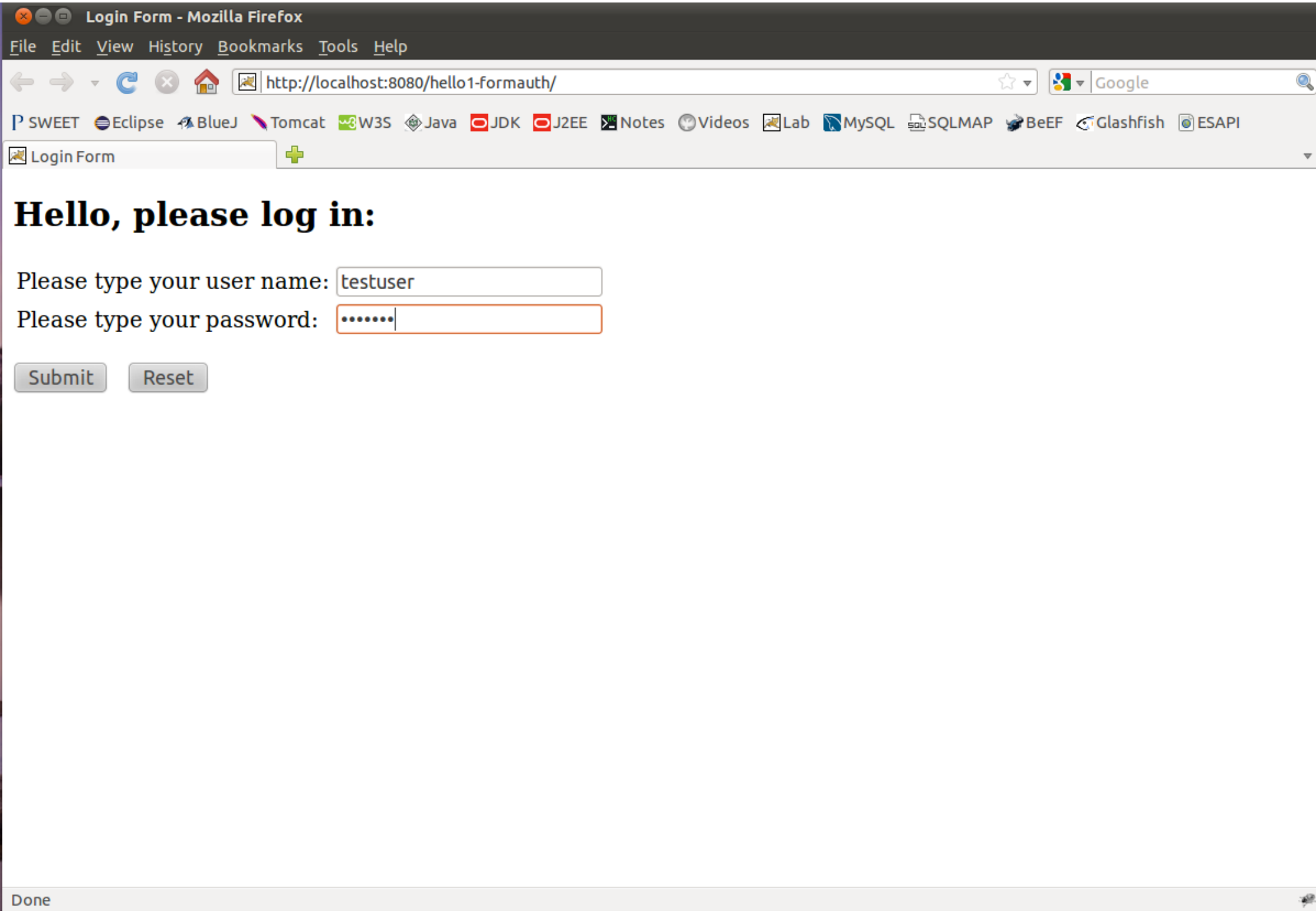
Figure 25–7 Form-Based Login Page



2. Type a user name and password combination that corresponds to a user who has already been created in the `file` realm of the GlassFish Server and has been assigned to the group of `TutorialUser` .

   Form-based authentication is case sensitive for both the user name and password, so type the user name and password exactly as defined for the GlassFish Server.


Login Form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello1-formauth/
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Login Form
Hello, please log in:
Please type your user name:
Please type your password:
Submit
Reset
Done


Login Form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello1-formauth/
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Login Form
Hello, please log in:
Please type your user name: testuser
Please type your password:
Submit
Reset
Done

Facelets Hello Greeting - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:8080/hello1-formauth/

Google

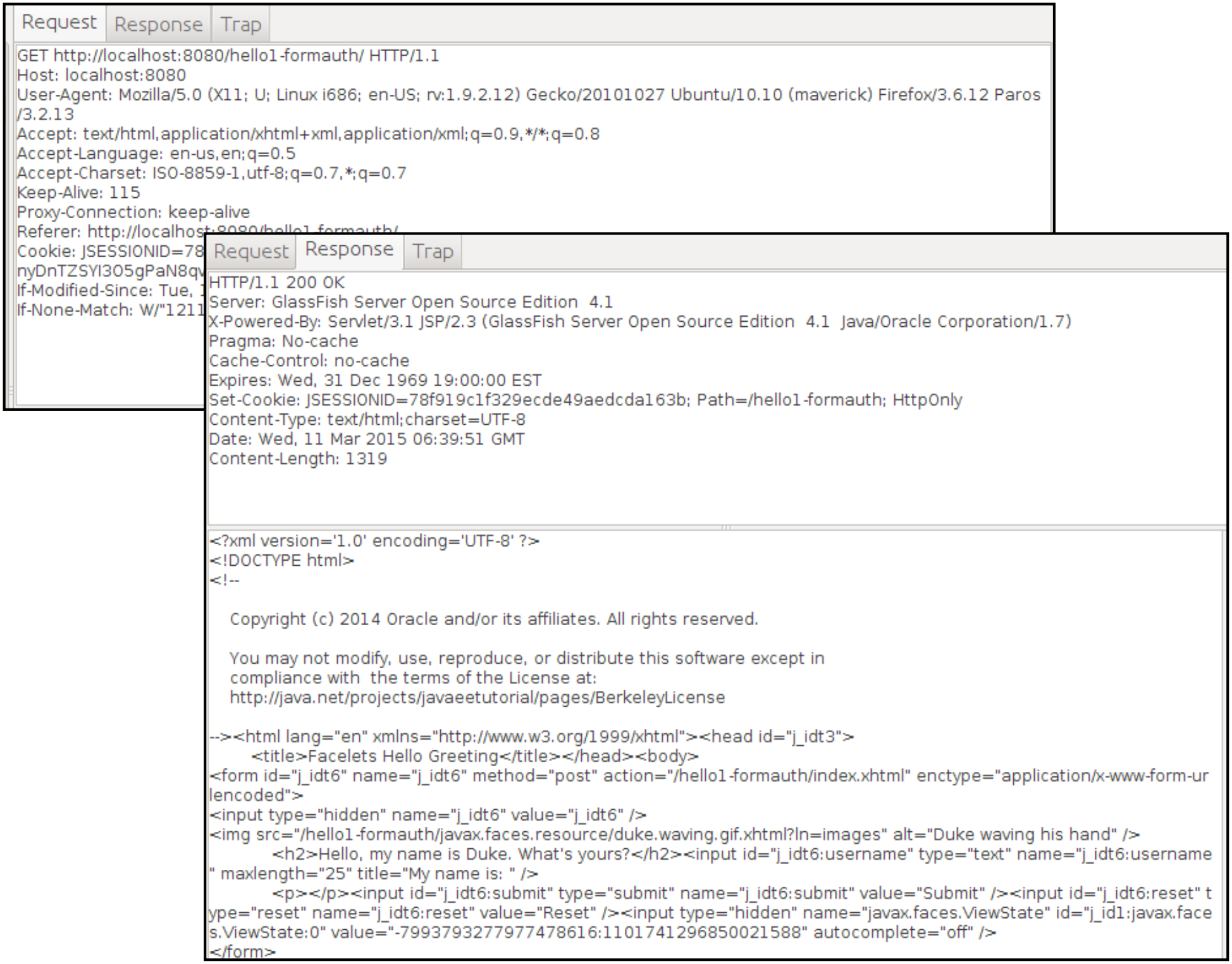
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI

Facelets Hello Greeting

Hello, my name is Duke. What's yours?

Submit Reset

Done



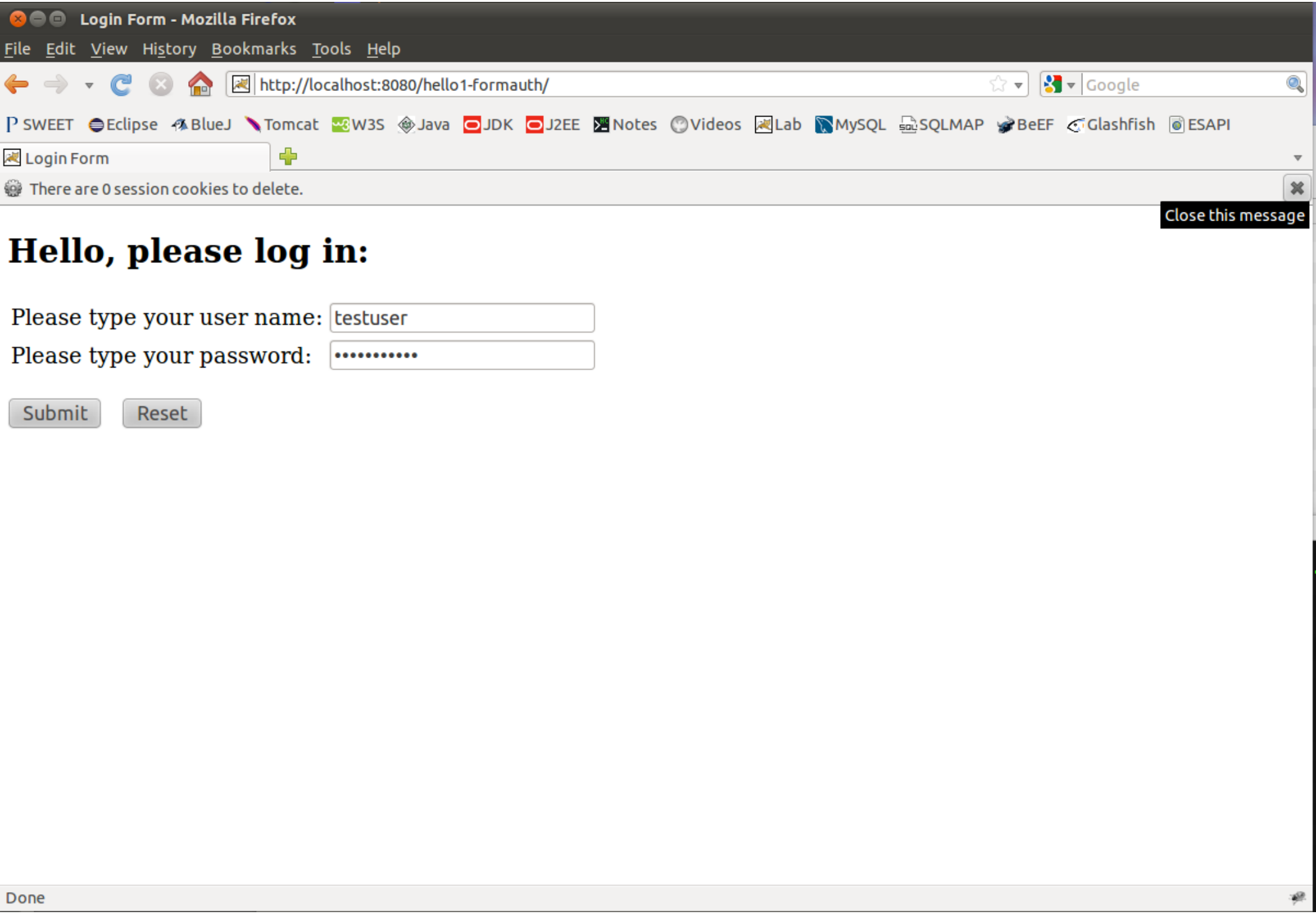
Request | Response | Trap

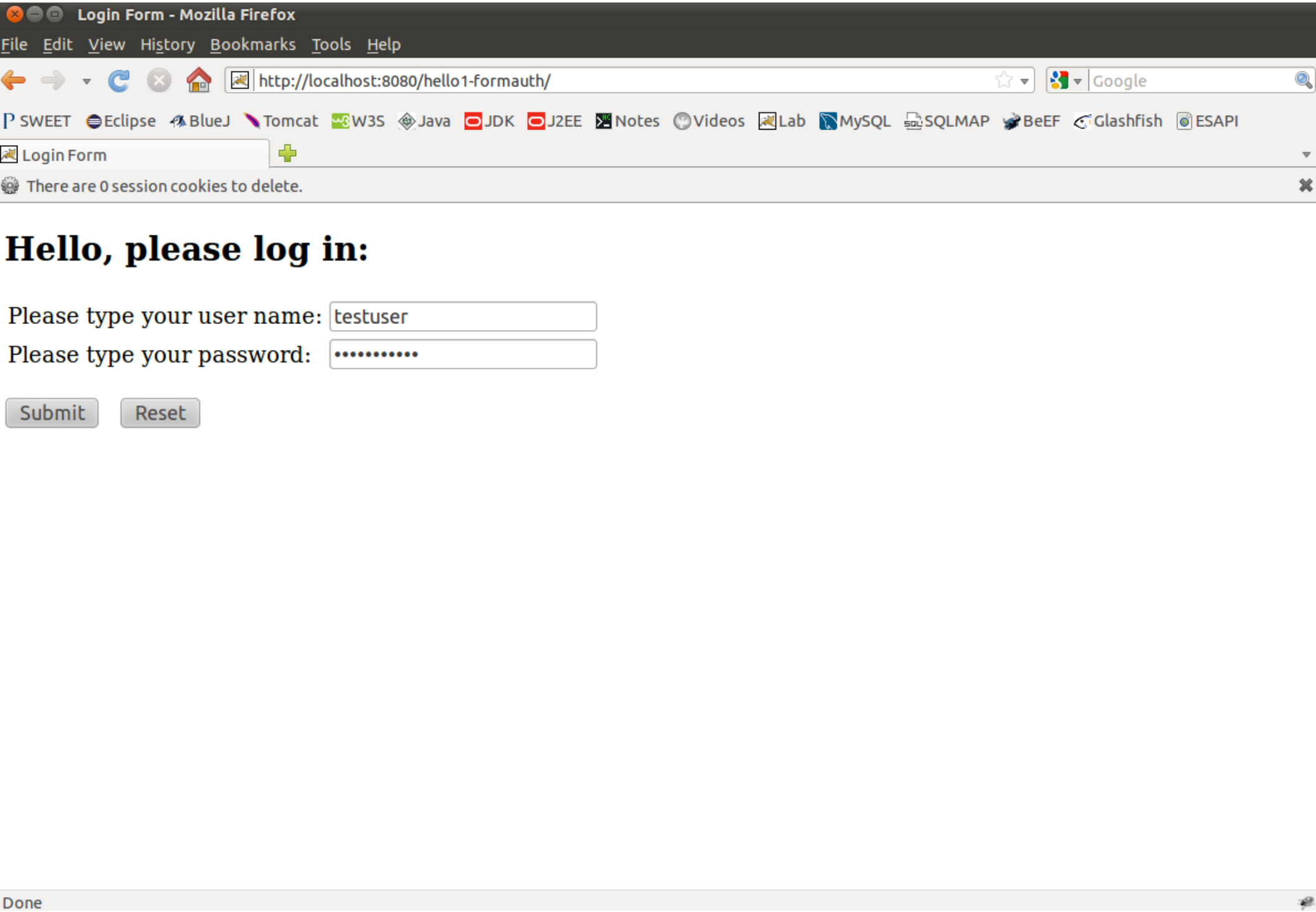
```
GET http://localhost:8080/hello1-formauth/ HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.12) Gecko/20101027 Ubuntu/10.10 (maverick) Firefox/3.6.12 Paros
/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
Referer: http://localhost:8080/hello1-formauth/
Cookie: JSESSIONID=78
nyDnTZSYI3O5gPaN8qv
If-Modified-Since: Tue,
If-None-Match: W/"1211
```

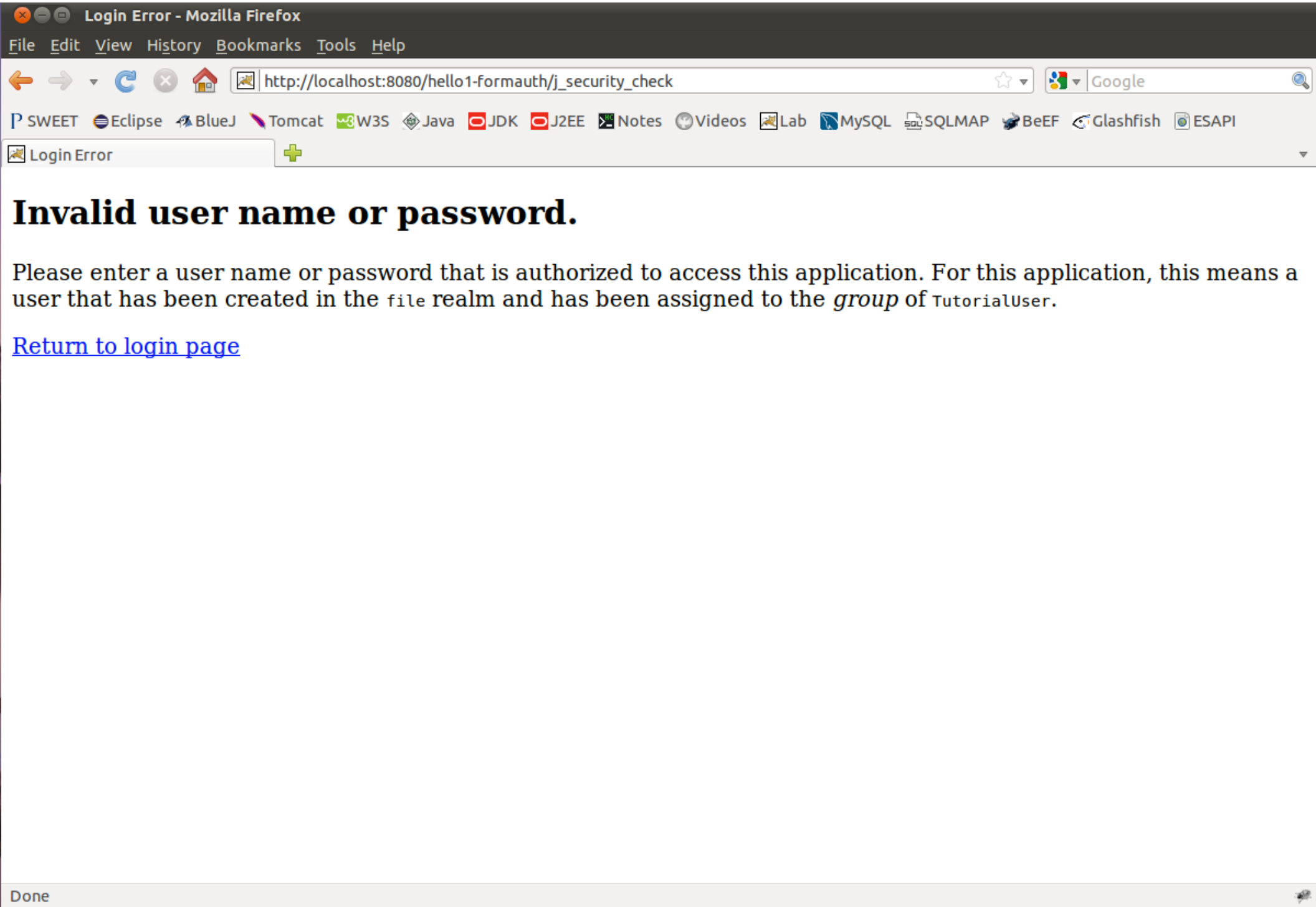
Request | Response | Trap

```
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/Oracle Corporation/1.7)
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 19:00:00 EST
Set-Cookie: JSESSIONID=78f919c1f329ecde49aedcda163b; Path=/hello1-formauth; HttpOnly
Content-Type: text/html;charset=UTF-8
Date: Wed, 11 Mar 2015 06:39:51 GMT
Content-Length: 1319
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<!--

    Copyright (c) 2014 Oracle and/or its affiliates. All rights reserved.

    You may not modify, use, reproduce, or distribute this software except in
    compliance with  the terms of the License at:
    http://java.net/projects/javaeetutorial/pages/BerkeleyLicense

--><html lang="en" xmlns="http://www.w3.org/1999/xhtml"><head id="j_idt3">
        <title>Facelets Hello Greeting</title></head><body>
<form id="j_idt6" name="j_idt6" method="post" action="/hello1-formauth/index.xhtml" enctype="application/x-www-form-ur
lencoded">
<input type="hidden" name="j_idt6" value="j_idt6" />
<img src="/hello1-formauth/javax.faces.resource/duke.waving.gif.xhtml?ln=images" alt="Duke waving his hand" />
            <h2>Hello, my name is Duke. What's yours?</h2><input id="j_idt6:username" type="text" name="j_idt6:username
" maxlength="25" title="My name is: " />
            <p></p><input id="j_idt6:submit" type="submit" name="j_idt6:submit" value="Submit" /><input id="j_idt6:reset" t
ype="reset" name="j_idt6:reset" value="Reset" /><input type="hidden" name="javax.faces.ViewState" id="j_id1:javax.face
s.ViewState:0" value="-7993793277977478616:1101741296850021588" autocomplete="off" />
</form>
```


Login Form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello1-formauth/
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Login Form
There are 0 session cookies to delete.
Close this message
Hello, please log in:
Please type your user name: testuser
Please type your password:
Submit
Reset
Done


Login Form - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello1-formauth/
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Login Form
There are 0 session cookies to delete.
Hello, please log in:
Please type your user name: testuser
Please type your password:
Submit
Reset
Done


Login Error - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello1-formauth/j_security_check
Google
SWEET Eclipse BlueJ Tomcat W3S Java JDK J2EE Notes Videos Lab MySQL SQLMAP BeEF Glashfish ESAPI
Login Error
Invalid user name or password.
Please enter a user name or password that is authorized to access this application. For this application, this means a user that has been created in the file realm and has been assigned to the group of TutorialUser.
Return to login page
Done

Sites

- Sites
  - http://localhost:8080
    - hello1-formauth
      - POST:j_security_check(j_password,j_username)

Request | Response | Trap

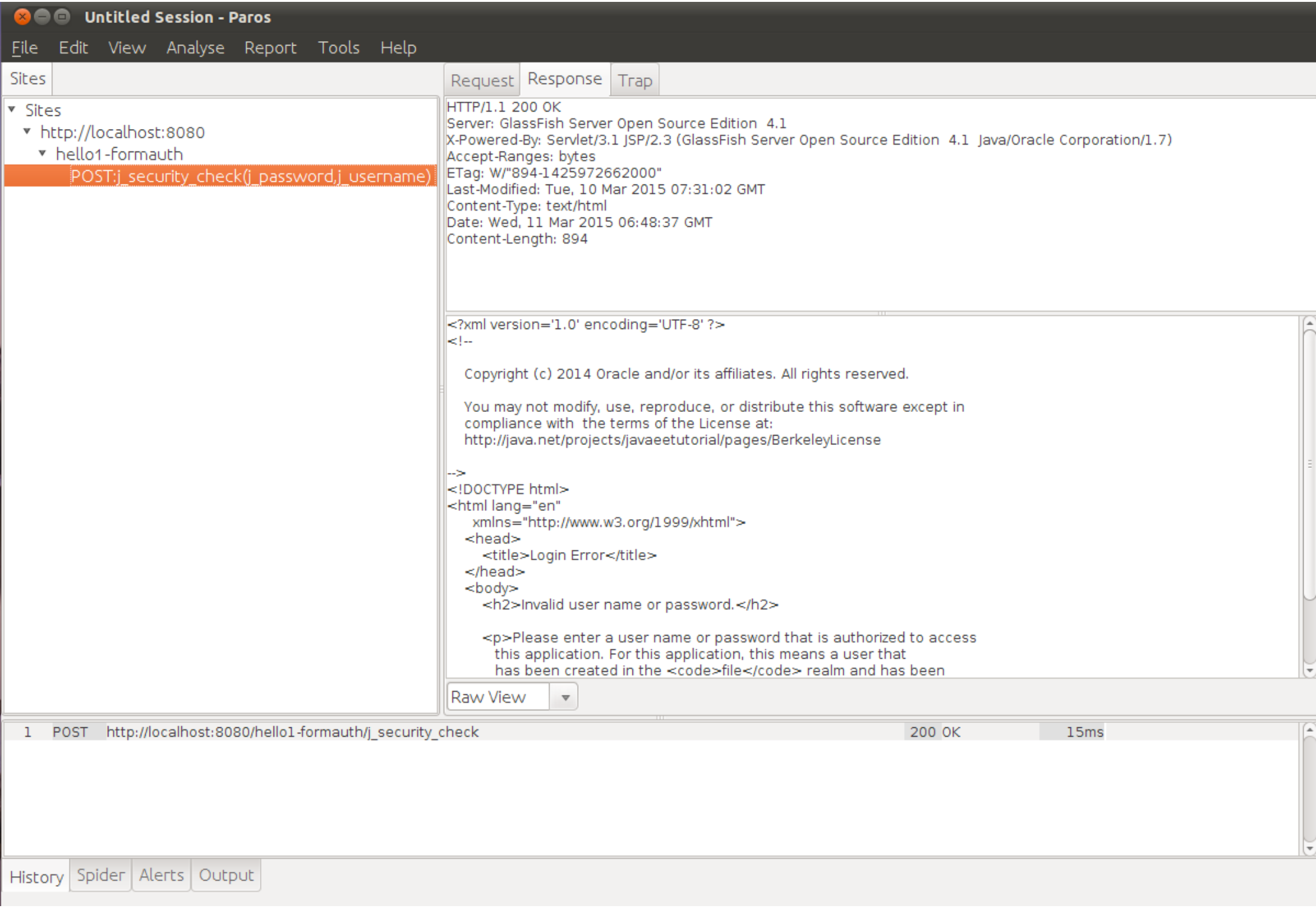
```
POST http://localhost:8080/hello1-formauth/j_security_check HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.12) Gecko/20101027 Ubuntu/10.10 (maverick) Firefox/3.6.12 Paros
/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
Referer: http://localhost:8080/hello1-formauth/
Cookie: JSESSIONID=796b8f38059200db83f255832ad8
Content-Type: application/x-www-form-urlencoded
Content-Length: 42
```

```
j_username=testuser&j_password=badpassword
```

Raw View

1	POST	http://localhost:8080/hello1-formauth/j_security_check	200 OK	15ms

History | Spider | Alerts | Output


Untitled Session - Paros
File Edit View Analyse Report Tools Help
Sites
Request Response Trap
Sites
http://localhost:8080
hello1-formauth
POST:j_security_check(j_password,j_username)
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition 4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.1 Java/Oracle Corporation/1.7)
Accept-Ranges: bytes
ETag: W/"894-1425972662000"
Last-Modified: Tue, 10 Mar 2015 07:31:02 GMT
Content-Type: text/html
Date: Wed, 11 Mar 2015 06:48:37 GMT
Content-Length: 894
<?xml version='1.0' encoding='UTF-8' ?>
<!--
Copyright (c) 2014 Oracle and/or its affiliates. All rights reserved.
You may not modify, use, reproduce, or distribute this software except in
compliance with the terms of the License at:
http://java.net/projects/javaeetutorial/pages/BerkeleyLicense
-->
<!DOCTYPE html>
<html lang="en"
xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Login Error</title>
</head>
<body>
<h2>Invalid user name or password.</h2>
<p>Please enter a user name or password that is authorized to access
this application. For this application, this means a user that
has been created in the <code>file</code> realm and has been
Raw View
1 POST http://localhost:8080/hello1-formauth/j_security_check 200 OK 15ms
History Spider Alerts Output

OWASP Enterprise Security API (ESAPI)

Log in Request account

Category Discussion Read View source View history Search

Category:OWASP Enterprise Security API

Home Downloads What I did with ESAPI Glossary Java EE Project Details [edit]

ESAPI (The OWASP Enterprise Security API) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications. The ESAPI libraries are designed to make it easier for programmers to retrofit security into existing applications. The ESAPI libraries also serve as a solid foundation for new development.

Allowing for language-specific differences, all OWASP ESAPI versions have the same basic design:

- **There is a set of security control interfaces.** They define for example types of parameters that are passed to types of security controls.
- **There is a reference implementation for each security control.** The logic is not organization-specific and the logic is not application-specific. An example: string-based input validation.
- **There are optionally your own implementations for each security control.** There may be application logic contained in these classes which may be developed by or for your organization. An example: enterprise authentication.

This project source code is licensed under the BSD license, which is very permissive and about as close to public domain as is possible. The project documentation is licensed under the Creative Commons license. You can use or modify ESAPI however you want, even include it in commercial

Let's talk here

ESAPI Communities

Further development of ESAPI occurs through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. For more information, please subscribe to one of the lists below.

- esapi-dev mailing list (this is the main list)
- esapi-user mailing list
- esapi-php mailing list
- esapi-python mailing list
- esapi-ruby mailing list
- esapi-swingset mailing list
- esapi-coldfusion mailing list

IRC Chat

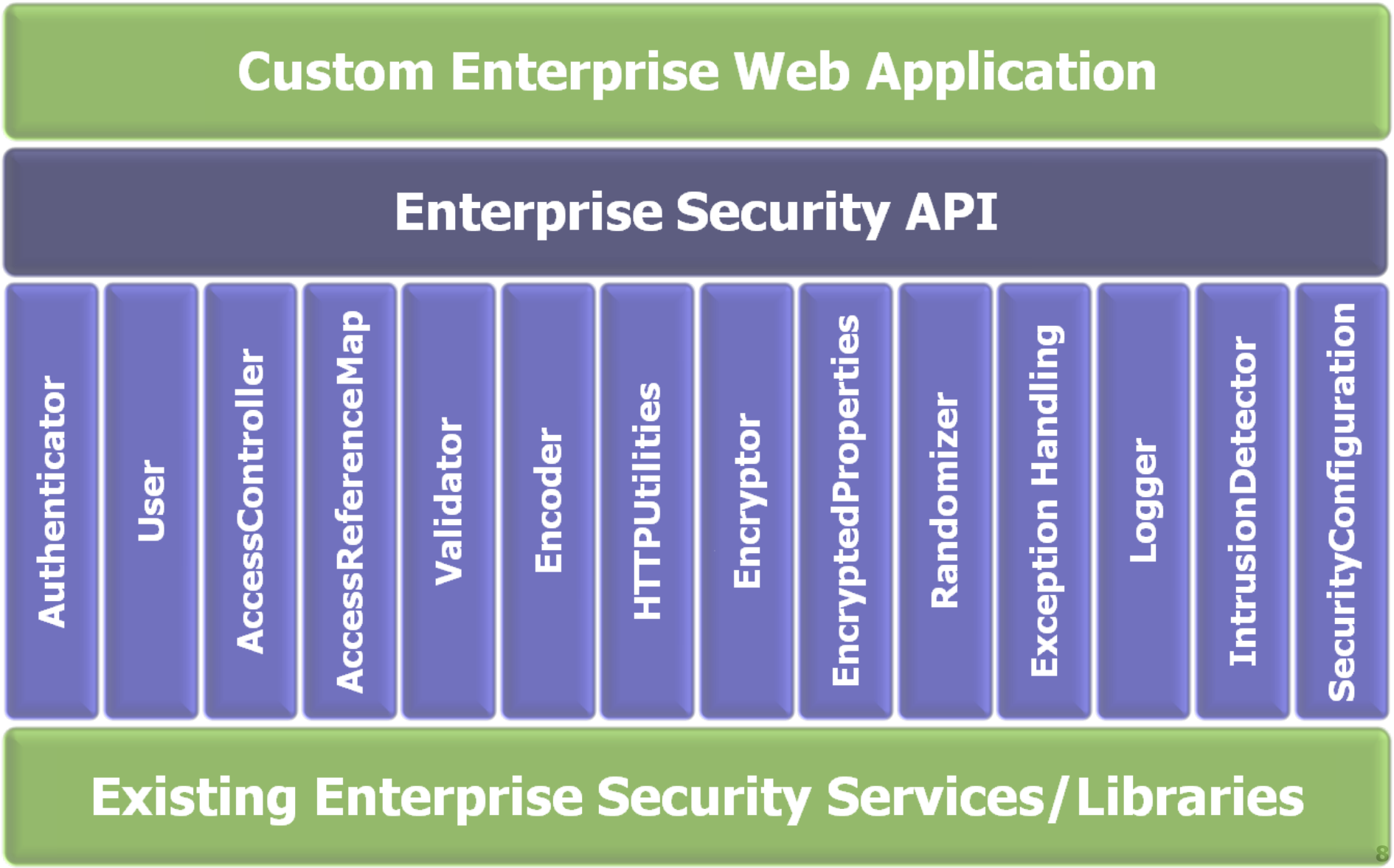
If you would rather chat with us about your problem or thoughts - you can join us in our IRC

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal
Presentations
Press
Projects
Video
Volunteer

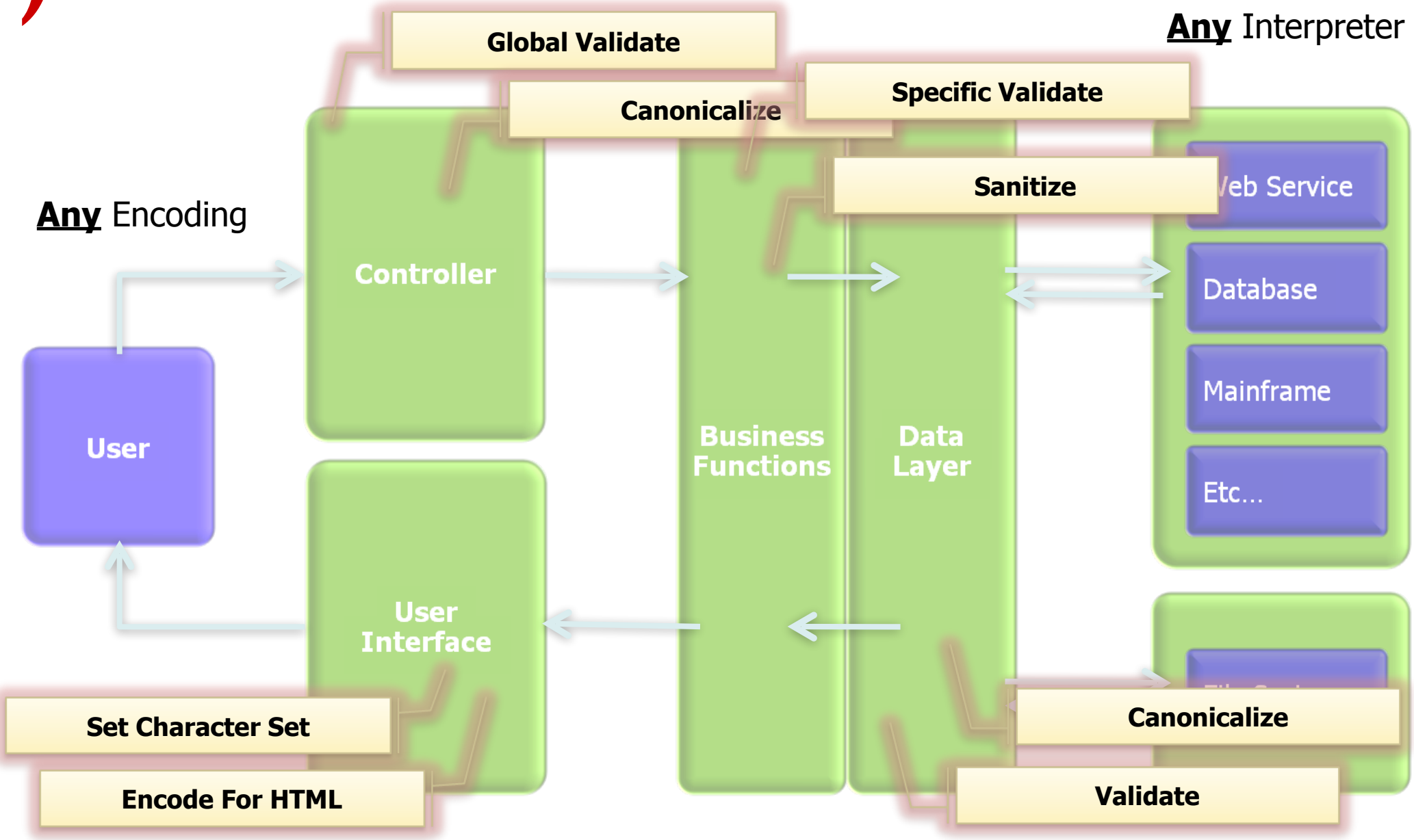
Reference
Activities
Attacks
Code Snippets
Controls
Glossary
How To...
Java Project
.NET Project
Principles
Technologies
Threat Agents

https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API#tab=Home

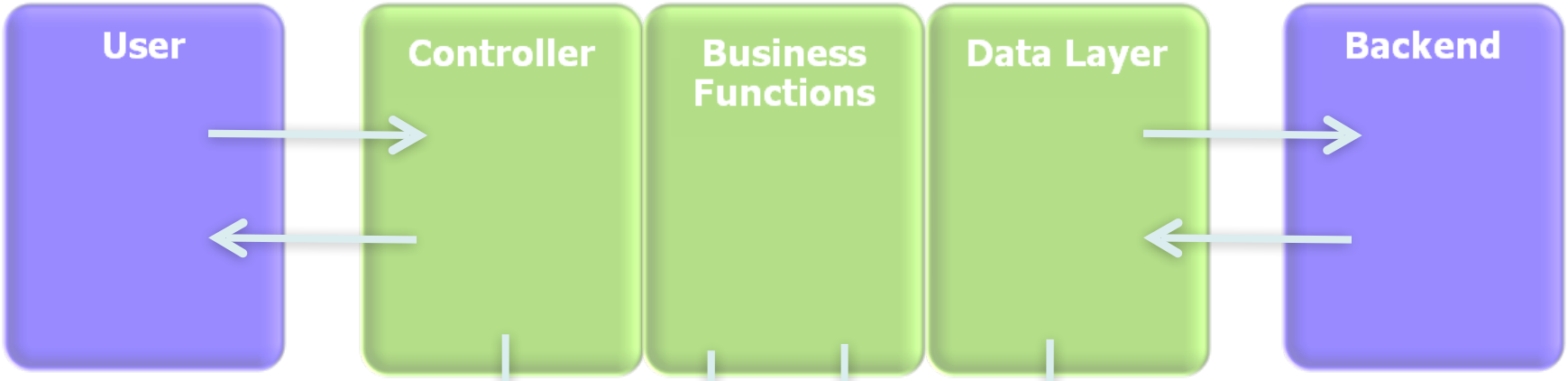
Enterprise Security API



Validation, Encoding, and Injection



Handling Validation, and Encoding

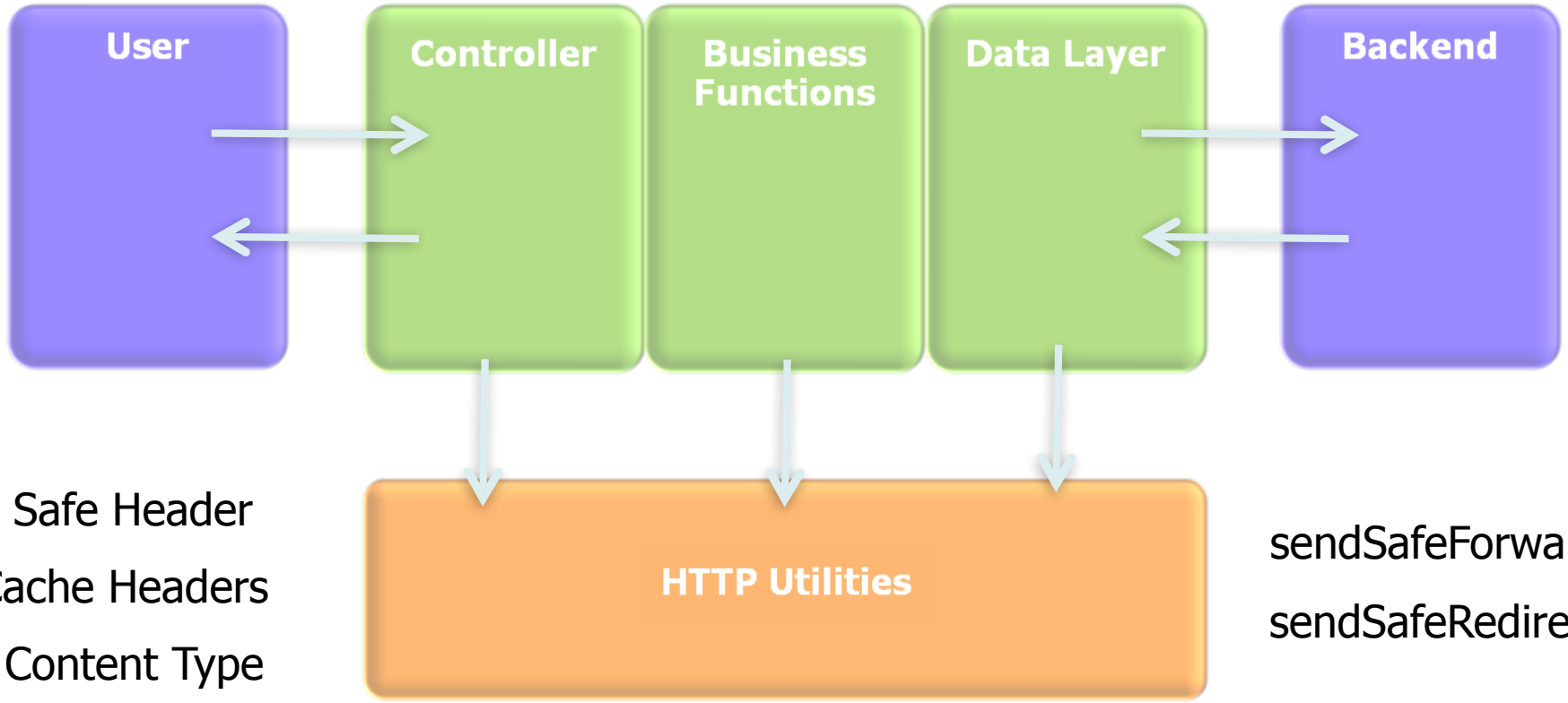


isValidCreditCard
isValidDataFromBrowser
isValidDirectoryPath
isValidFileContent
isValidFileName
isValidHTTPRequest
isValidListItem
isValidRedirectLocation
isValidSafeHTML
isValidPrintable
safeReadLine

Canonicalization
Double Encoding Protection
Sanitization
Normalization

encodeForJavaScript
encodeForVBScript
encodeForURL
encodeForHTML
encodeForHTMLAttribute
encodeForLDAP
encodeForDN
encodeForSQL
encodeForXML
encodeForXMLAttribute
encodeForXPath

Handling HTTP



Add Safe Header

No Cache Headers

Set Content Type

Add Safe Cookie

Kill Cookie

Change SessionID

CSRF Tokens

isSecureChannel

Safe Request Logging

Safe File Uploads

sendSafeForward

sendSafeRedirect

Encrypt State in Cookie

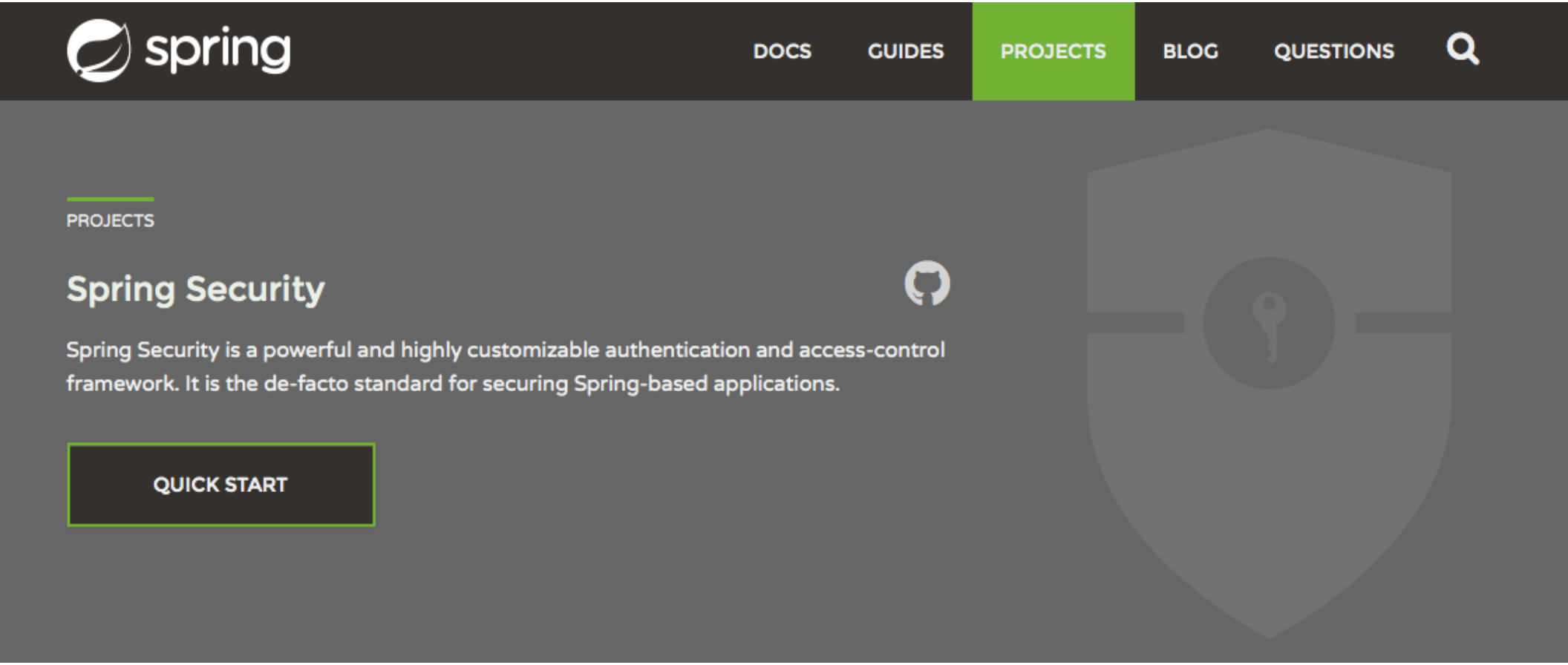
Hidden Field Encryption

Querystring Encryption

Coverage

OWASP Top Ten	OWASP ESAPI
A1. Cross Site Scripting (XSS)	Validator, Encoder
A2. Injection Flaws	Encoder
A3. Malicious File Execution	HTTPUtilities (Safe Upload)
A4. Insecure Direct Object Reference	AccessReferenceMap, AccessController
A5. Cross Site Request Forgery (CSRF)	User (CSRF Token)
A6. Leakage and Improper Error Handling	EnterpriseSecurityException, HTTPUtils
A7. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A8. Insecure Cryptographic Storage	Encryptor
A9. Insecure Communications	HTTPUtilities (Secure Cookie, Channel)
A10. Failure to Restrict URL Access	AccessController

Spring Security



http://projects.spring.io/spring-security/