Objectives

- JSTL Review
- Designing for Security

JSTL Review

- What is the purpose of JSTL?
  - What is the goal of JSTL?
  - What is JSTL?
- What are some common tags in the JSTL Core library?
  - How do they work?
  - What do they do?

Fixing Our JSTL Problem

- `cookie` is a HashMap
- Wrong: HashMap.Entry object has no name

```jsp
<h1>Cookies: </h1>
<c:forEach var="entry" items="${cookie}">
  ${entry} - ${entry.name}<br />
</c:forEach>
```

- Right: Get the Cookie object out of the HashMap.Entry object

```jsp
<h1>Cookies: </h1>
<c:forEach var="entry" items="${cookie}">
  ${entry.value} - ${entry.value.name} <br />
</c:forEach>
```

Security

“A few lines of code can wreak more havoc than a bomb.”

-- Tom Ridge
(Former) Secretary of the U.S. Department of Homeland Security

Web Security

- Why is the Web a target?

Web Application Architecture

“75% of cyber attacks and Internet security violations are generated through Internet Applications” - Gartner Group

From www.itsa.ufl.edu/2006/presentations/malpani.ppt
Web Security

- Why is the Web a target?
  - New environment: lots of vulnerabilities
  - New technologies to exploit
  - Many different targets
    - Web browsers, Web servers, databases, personal computers, network traffic, humans
  - Efficient, lucrative
    - Can get a lot done quickly
    - Personal info, credit card info, databases
    - Businesses: could lose millions of dollars/hour

- What are some Web vulnerabilities?

In the News

- May 22 in Yahoo! News: LiveLock CEO’s identity stolen

Potential Infections

- Virus
  - Attached to a host program
  - Attack infected computer
- Worm
  - Self-replicated computer program
  - Attack network, consume bandwidth
- Backdoor
  - Bypass normal authentication
- Trojan horse
  - Perceived “good” program that does bad
  - Could install backdoor programs

Potential Infections

- Malicious Code Types
  - Virus
    - Attached to a host program
    - Attack infected computer
  - Worm
    - Self-replicated computer program
    - Attack network, consume bandwidth
  - Backdoor
    - Bypass normal authentication
  - Trojan horse
    - Perceived “good” program that does bad
    - Could install backdoor programs

OWASP Top 10 Vulnerabilities

- Unvalidated input
- Broken Access Control
- Cross Site Scripting (XSS) Flaws
- Buffer Overflows
- Injection Flaws
- Insecure Error Handling
- Insecure Storage
- Denial of service
- Insecure Configuration Management

Security Requirements

- Authentication
  - Verify the user’s identity
- Authorization
  - Give access to resources only to allowed users
- Confidentiality
  - Ensure that information is passed only to allowed users
- Integrity
  - Ensure that data is not inappropriately changed
Security Design Principles

- Least Privilege
- Defense in Depth
- Secure Weakest Link
- Fail-safe Stance
- Secure By Default
- Simplicity
- Usability

Principle of Least Privilege

- Give just enough permissions to get the job done
  - Minimize damage if compromised
- Common world example: Valet Keys
  - Can start the car but can’t access glove compartment, trunk
- A web server should only be given access to the set of HTML files that the web server is to serve
  - If server is compromised, attacker can only read HTML files

Defense in Depth

- Also called redundancy / diversity
- Common world example: Banks
  - What security does a bank provide so that bad guys won’t steal money?
- Passwords:
  - Require users to choose “strong” passwords
  - Monitor web server logs for failed login attempts
  - If 3 incorrect password attempts, lock account for some period of time

Secure the Weakest Link

- Common Weak Links:
  - Unsecured Dial-In Hosts - War Dialers
  - Weak Passwords - Crack
  - People - Social Engineering Attacks
    - “I am owed an inheritance; you can get a cut if you just give me your bank account number…”
  - Buffer Overflows
    - Slapper Worm: exploited OpenSSL, Apache to create peer-to-peer networks
    - Not a problem in Java
    - More on this later...

Fail-Safe Stance

- Even if 1 or more components of system fail, there is some security
- Common world example: Elevators
  - If elevator power fails, grip the cable
- System failure should be expected (and planned for)
  - If firewall fails, let no traffic in
  - Deny access by default

Secure By Default

- “Hardening” a system: All unnecessary services off by default
  - Only enable the 20% of the products features that are used by 80% of the user population
  - Ex: don’t enable insecure Windows’ networking apps by default
  - If choose to use those apps, know what you’re doing
- More features enabled ➔ more potential exploits ➔ less security!
**Simplicity**
- Complex software is more likely to have security holes
  - More code that wasn’t tested thoroughly
  - More opportunities for buffer overflows, etc.
- Keep security checks localized with **choke points**: centralized piece of code through which control must pass
- Generally: good rule of thumb to keep software simple

**Usability**
- Users typically do not read documentation
  - Enable security by default
- Users can be lazy
  - Assume: They ignore security dialogs
- **Secure by default features in software** forces users and vendors to be secure
- More on usability in a bit…

**Applying Security**
- Can be applied at multiple levels:
  - Web server (e.g., Apache)
  - Servlet container (e.g., Tomcat, Resin)
  - Web application
- Each level offers different amounts of security and flexibility

**Security Application Levels**
- **Web server**
  - HTTP authentication
  - Authorization of users/groups
  - Authorization of domains
  - HTTPS: a secure version of HTTP
- **Servlet container**
  - HTTP authentication (basic, digest)
  - Form-based authentication
  - Authorization of users/groups
  - SSL capabilities
- **Application**
  - Authorization of users
  - User information kept on server in a session

**Apache’s User-Level HTTP Passwords**
- Protect a Web-accessible directory with `.`htaccess file
  - Goes into the directory you want to protect
  - Special format
- Requires users to enter a username, password to access files in the directory
- Can also create authorized **groups**

**Discussion of Passwords**
- What makes a good password?
  - To the software?
  - To a user?
- How do passwords, their restrictions affect software?
  - How do they affect those who try to break them?
- How often should users change their passwords?
Security vs. Usability

- Hard to design passwords that are both secure and easy to remember
- If users have to change their password more than every 6 months, security decreases
  - Come up with schemes to make passwords easier to remember

Apache: Access Control

- Allow or deny requests by their domain
  - Syntax: \{allow/deny\} from address
    - deny from 11.22.33.44
    - deny from hostname
    - deny from 192.101.205
    - deny from exampleone.com exampletwo.com

Secure Socket Layer (SSL)

- Encrypt every HTTP message to and from the web server using standard PKI (public key infrastructure) technology
- De-facto standard used for secure web-based transactions
- Default URL https://some.domain.com with default port number 443

Application: Careful Exception Handling

- Effective exception handling is essential in designing for security from the start

Careful Exception Handling

- Error messages and observable behavior can tip off an attacker to vulnerabilities
  - Be careful not to provide more information to a user than is needed
- Exploit vulnerabilities using Fault Injection
  - Provide an application with input that it does not expect and observe its behavior
  - Bad guys: Use this to figure out application’s vulnerabilities
    - Sever may have fatal exception

Error Pages

- What you don’t want
  - Show the Server’s default exception page
  - Show the stack track
  - Looks like your application is broken to user
- Want error page to look like the rest of your site
JSP Error Pages

- Defining error pages in deployment descriptor (web.xml)
  
  ```xml
  <error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error-page.jsp</location>
  </error-page>
  ```
- A JSP Error Page includes
  
  ```xml
  <%@ page isErrorPage="true" %>
  ```
- A JSP page can define its error page
  
  ```xml
  <%@ page errorPage="errorPage.jsp" %>
  ```
- Can have user-defined exceptions
- Can be a path to a servlet

Common Attacks & Vulnerabilities

Denial of Service (DoS)

- An attacker makes a web server unavailable
- Typically, server (or other resource) is flooded with malicious requests
  - Legitimate requests can’t get through or the response time is too slow to matter
- Variations
  - Mail bomb: spam attack on mail server
  - DDoS: multiple clients
    - Could be willing participants or compromised servers

Input Validation Vulnerabilities

- Caused by insufficient input validation
  - Using unvalidated input can cause errors
- Common attacks: Buffer overflow, Cross site scripting, SQL injection

Cross-Site Scripting (XSS)

- Cross-site scripting is possible when
  - An adversary trick a victim into clicking a link crafted and presented to the victim via a web server or email
  - The link contains a URL with embedded malicious script (typically as a query string, for example “phishing”)
  - The URL refers to host that echoes input back to a browser without input validation
- When victim clicks link, goes to the host in the URL
  - Host processes the query string, echoes it to victim’s browser
- Victim’s browser executes the malicious script
- Root Cause: Failure to proactively reject or scrub malicious characters from input vectors

Buffer Overflows

- An error caused when a program tries to store too much data in a temporary storage area
  - Exploited by hackers to execute malicious code

Data from 2006

From Web Application Security Consortium
Cross-Site Scripting (XSS)

- Allows cookie theft, credential theft, data confidentiality, integrity, and availability risks
  - Browser Hijacking and Unauthorized Access to Web Application is also possible using existing exploits
- Unusual vulnerability because the system at fault, i.e., the web site not validating input, is not the victim of attack
- Remedy for XSS: web site perform **adequate input validation**
  - Global policy, Form- and Field- specific policies for handling untrusted content

Sequence Diagram of a XSS Attack

```
Attacker

Email with link to web site

Victim

Vulnerable Web Site

Victim navigates to web site

Attacker has access in victim's context

Malicious script runs
```

Unvalidated Input with XSS

From www.itsa.ufl.edu/2006/presentations/malpani.ppt

Unvalidated Input resulted in a Cross-Site Scripting Attack and theft of Administrator’s Cookie

From www.itsa.ufl.edu/2006/presentations/malpani.ppt

Unvalidated Input with XSS

From www.itsa.ufl.edu/2006/presentations/malpani.ppt

Unvalidated Input with XSS

From www.itsa.ufl.edu/2006/presentations/malpani.ppt
Cross-Site Scripting: Content Spoofing

- Insert un-trusted content into the web application that can be used to trick users
- Compromise the integrity of application code via malicious script code injected into the database
- Limited only by the attackers’ imagination

Cross-Site Scripting Exploit

```html
<script>
var oWH = window.open("", "width=275, height=175, top=200, left=250 location=no, menubar=no, status=no, toolbar=no, scrollbars=no, resizable=no");
oWH.document.write("HTML FORM with POST request to http://compromised-server/h4xor.php");
</script>
```

XSS: Content Spoofing

![XSS: Content Spoofing](http://www.itsa.ufl.edu/2006/presentations/malpani.png)

Stored XSS

![Stored XSS](http://www.itsa.ufl.edu/2006/presentations/malpani.png)

Testing for XSS

- Test for valid HTML and script code allowed in an input field
  - Special characters like `<` or `>`
  - `<script>alert("XSS");</script>`
  - `<script>alert(document.cookie);</script>`
  - `article.php?title=<meta%20http-equiv="refresh"%20content="0;">`
- Causes denial of service
- References:
  - http://ha.ckers.org/xss.html

SQL Injection

- Possible vulnerability when a program accepts unvalidated input from a user and uses that input to construct a dynamic SQL query to an SQL database
  - Client may construct crafted input that, when embedded in a string, is interpreted as an SQL query
  - Performs database operations not intended by application writers
SQL Injection
• Possible vulnerability when a program accepts unvalidated input from a user and uses that input to construct a dynamic SQL query to an SQL database
  ➢ Client may construct crafted input that, when embedded in a string, is interpreted as an SQL query
  ➢ Performs database operations not intended by application writers

SQL Injection
• Root Cause: Failure to properly scrub, reject, or escape domain-specific SQL characters from an input vector
• Solution:
  ➢ Define accepted character-sets for input vectors, and enforce these white lists rigorously.
  ➢ Force input to conform to specific patterns when other special characters are needed: dd-mm-yyyy
  ➢ Use SQL Prepared Statements

SQL Injection
• Typical query to email forgotten password:

```
SELECT fieldlist
FROM table
WHERE field = '$EMAIL';
```

• Is the input sanitized? Try sprenkles@wlu.edu

```
SELECT fieldlist
FROM table
WHERE field = 'sprenkles@wlu';
```

➢ If not, the query will throw exception

SQL Injection
• How to exploit:

```
SELECT fieldlist
FROM table
WHERE field = 'anything' OR 'x'='x';
```

➢ Query expected to only return one entry
  ➢ This one will return all entries in user table
  ➢ Probably only displays the first response

➢ Can start to guess columns in table and table's name

Example: SQL Tautology Injection
Submitting SQL Query logic instead of a valid date can expose confidential records.

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x' AND members.email IS NULL; --';
```

➢ Don't need to worry about matching quotes

➢ Is database read-only?

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x'; DROP TABLE members; --';
```

Example: SQL Tautology Injection
Unvalidated Input allows SQL Injection

From www.itsa.ufl.edu/2006/presentations/malpani.ppt
Example: SQL Tautology Injection

Submitting SQL Query logic instead of a valid date can expose confidential records.

Validating Input

- **Black list**: a list of input types that are expressly forbidden from being used as application input
- **White list**: a list of input types that are expressly allowed from being used as application input
- Generally expressed as **regular expressions**
- Input validation must be server side
  - Not in JavaScript

Security Features Do Not Imply Security

- Using one or more security algorithms/protocols will not solve all your problems!
  - Using encryption doesn’t protect against weak passwords.
  - Using SSL in a Web server doesn’t protect against DoS attacks, access to various files, etc.

“Good Enough” Security

- The fraction of time you spend designing for security in your application should be proportional to the number and types of threats that your software and business face
- Remember: Customers expect privacy and security
- **Design for security by incorporating “hooks” and other low-effort functionality from the beginning**
  - Add more security as needed without having to resort to work-arounds.

Don’t Reinvent the Wheel

- Building a secure, high-performance web server is a very challenging task
  - Apache: www.apache.org
- Use trusted components
  - Keep up-to-date with security patches
Where Are Our Security Vulnerabilities?

TODO

- Project
  - Final implementation deadline: Fri
  - Documentation: Monday
  - Input validation, PreparedStatements