Web Application Security

SIRT Security Training
April 9th, 2009
Introduction

Philip Sears
Application Development Manager
Office of Mediated Education
Kansas State University

- Technical Lead on K-State Online/Axio
- About 9 years Java EE development experience at K-State.
Technology Background

• Oracle Database (9i -10g)
• JBoss Application Server (4.0.4 – 4.2.3)
• Java EE (Enterprise Edition)
  • EJB(2.1 – 3.0) (back-end, data tier)
  • MVC frameworks:
    • Struts, Struts2, JSF, WebExec
• JSP (Java Server Pages)
• HTML/CSS/JavaScript
Web Application Security Topics

1. Encryption
2. Authentication
3. Authorization
4. Input Validation
5. Internal Details
6. Error Handling
Encryption

• SSL (Secure Sockets Layer)
  – HTTPS:// (Secure-HTTP)

• SSL Certificates
  – Encryption (128-bit?)
  – Identification - Proves this site is who it says it is
    • Certificate issued by Certificate Authority
    • Prevents phishing attacks

• HTTPS, Always.
Authentication

- Authentication
  - Determine whether someone is actually who they declared to be.

- Methods
  - Form Based, Basic, Integrated, Certificate, etc

- Single Sign On

- LDAP – (Lightweight Directory Access Protocol)

- Authenticate all sites not deliberately public.
Authentication

- **HttpSession** allows users info to persist across page request

- **Session ID**
  - Stored in **cookie**, URL, hidden input
  - Predictability - Length/Sequential/Hashing
  - Session Hijacking

- **Secure Transmission**
  - HTTPS, once again

- **Length of Session Validity**
  - Server should timeout these HttpSession
  - Logout - `httpSession.invalidate()`
<table>
<thead>
<tr>
<th>NAME</th>
<th>JSESSIONID</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE</td>
<td>A16EBD69B558C6B5F3DC6A0C0A61F38E.ome-as2--axio</td>
</tr>
<tr>
<td>HOST</td>
<td>online.ksu.edu</td>
</tr>
<tr>
<td>PATH</td>
<td>/</td>
</tr>
<tr>
<td>SECURE</td>
<td>Yes</td>
</tr>
<tr>
<td>EXPIRES</td>
<td>At End Of Session</td>
</tr>
</tbody>
</table>
Authorization

• **Authorization**
  - *The process of giving someone permission to do or have something.*

• **Group / Role Based**
  - LDAP, AuthZ, AdminConsole

• **Access Control by Application**
  - Centralize/Reuse
  - Authorization Matrix
  - Importance of QA/Testing

• **Limit direct access to production environment.**
In WEB-INF/web.xml:

```xml
<filter-mapping>
  <filter-name>AdminFilter</filter-name>
  <url-pattern>/admin/*</url-pattern>
</filter-mapping>
```

```xml
<filter>
  <filter-name>AdminFilter</filter-name>
  <filter-class>teval.web.AdminFilter</filter-class>
</filter>
```

In Filter.java:

```java
public class AdminFilter implements Filter {
  public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) {
    String userName = req.getRemoteUser().trim().toUpperCase();
    TevalUser user = Services.getSecurityLib().getUser(userName);
    if (!user.getIsProgrammer() &amp;&amp; !user.getCanAdminSystem()) {
      throw new SecurityException("User "+userName+" cannot access admin tools ");
    }
  }
}
```

**Authorization Example - URL Filters**
Authorization Example – Redundancy

Front End
In Message.jsp:

```
<c:if test="${messageBoard.permissions.canModerate}"
    <a href="delete();">Delete Message</a>
</c:if>
```

Back End
In MbLibEJB.java:

```
public void deleteMessage(int messageId, CourseUser user) throws SecurityException {
    if (getSecurity(user, groupId).getCanModerate() ) {
        message.setDeleted(true);
    } else {
        throw new SecurityException("Not moderator of message " + messageId);
    }
}
```
Input Validation

• Invalid Input is #1 threat
  – Buffer Overflows, SQL Injection, Cross Site Scripting (XSS), Code Injection

• Preventative Measures
  – Form Validation
    • Strongly Typed, Length, Range, Syntax
    • MVC Frameworks (Struts)
  – SQL Injection example…
## Example of SQL Injection prevention: PreparedStatement

```java
// user enters "joe'; drop table users;" into input
<form><input type="text" name="username"></form>

// value received on server side
String userName = request.getParameter("username");

// Without PreparedStatement
Statement stmt = getConnection().createStatement(
    "select * from users where username = " + userName + ");
stmt.executeQuery();
// database executes select AND drops table:
- select * from users where username = 'joe';
- drop table users;

// WITH PreparedStatement
PreparedStatement stmt = getConnection().prepareStatement(
    "select * from users where username = ? ");
stmt.setString(1, userName);
stmt.executeQuery();
// database executes select using BOUND parameter(s)
- select * from users where username = $username
```
Input Validation

• Cross Site Scripting (XSS)
  – Attacks the end user and not your site while leveraging ‘trusted’ nature of your site.
  – Lets arbitrary code such as JavaScript run on clients computer.
  – Caused by allowing users to input code that will be rendered on a web page.

• Test Your Site:
  <script>alert("hello!");</script>
  <script src="http://www.malicious-host.com/badscript.js"></script>
  <script>alert(document.cookie);</script>
Input Validation

**Prevention - Filter Input**

- Remove JavaScript code & dangerous HTML tags:
  `<SCRIPT>, <EMBED>, <APPLET>, ONCLICK, ONLOAD, SRC`
- Sanitize all user input – (HTML entities)
  `<SCRIPT>` versus `&lt;SCRIPT&gt;`
- HTML Editors, Frameworks, etc

**Worst Case: Directly rendering parameters**

1. Url takes a parameter
   `/DisplayMessage?displayParam=paramValue`
2. Url directly renders parameter
   `<%= displayParam %>`
Input Validation Example - K-State eProfile

#1. Make sure that the request is not being submitted by external source:
   - Uses Struts tokens to prevent double submission.

#2. Make sure current user is real user by authentication
   - Requires the old password to be provided to change current password.
Hiding Internal Details

- Don’t show internal system details to users
  - No Stack Traces, SQL, Code, etc
- Comments Example:
  - `<%--- only for developers ☺ --%>`
  - `<!-- viewable by user in html ☹ -->`
- Relates to Error Handling
Error Handling

• Fail Safe
• Log Errors
• Java Exception Handling
  – Be Specific, Throw Early, Catch Late
    
    ```java
    try { // code throws } catch (Exception e) { // handle }
    ```

  – Checked vs Unchecked
  – Don’t use exceptions for normal functions
  – Don’t trap important exceptions:

    ```java
    catch (Exception e) { // doing nothing }
    ```

• Centralize Error Handling example…
// One ErrorHandler.java class to rule them all!
public class ErrorHandler implements ome.util.ErrorHandler {

    public String handleError(Commands lib, Throwable e) {
        // if the exception is wrapped get the true exception
        Throwable error = logError(lib, e)
        return getErrorJsp(error);
    }

    public static String getErrorJsp(Throwable e) {
        Throwable error = getSourceError(e);
        if (error instanceof SecurityException)
            return Constants.SECURITY_ERROR_JSP;
        return Constants.ERROR_JSP;
    }

    public static Throwable logError(Object source, Throwable e) {
        Throwable error = getSourceError(e);
        Logs.getLog(source).error("Error from "+req.getRemoteUser()+" at "+req.getRequestURL(), error);
        return error;
    }

    public static Throwable getSourceError(Throwable e) {
        while (true) {
            if (e instanceof InvocationTargetException) {
                e = ((InvocationTargetException)e).getTargetException(); continue;
            }
            if (e instanceof RemoteException) {
                e = ((RemoteException)e).detail; continue;
            }
            if (e instanceof JspException) {
                e = ((JspException)e).getRootCause(); continue;
            }
            break;
        }
        return e;
    }
}
SYSTEM ERROR

K-State Online has encountered a system error, please sign out and try again.

If you feel you have reached this message in error, or continue to receive this error, please contact IT Help Desk at (785) 532-7722, (800) 865-6143, or email: help@online.ksu.edu, or visit http://support.online.ksu.edu.

error.jsp – one generic error page

Simple HTML - Displays generic error message useful for users.
No stack trace or system details.
Error details can be found by programmers in system logs.
Other Security Topics

- Denial of Service
- Brute Force Attacks
- Buffer Overflow
- Config Management
- Secure Storage
- Network Security
- Email
Questions/Comments?

Philip Sears
phils@ksu.edu