An Introduction to Application Security in J2EE Environments

Dan Cornell
Denim Group, Ltd.
www.denimgroup.com

(c) Copyright 2005 Denim Group, Ltd.
Overview

- Background
- What is Application Security and Why is It Important?
- Specific Reference Examples
- J2EE Specific Examples
  - Access Controls
  - Input Validation
- Resources and Conclusions
Background

- Denim Group is a San Antonio based software development firm specializing in:
  - Custom software development
  - Business systems integration
  - Application-level security
  - J2EE and .NET environments

- Management team has breadth of experience working in and servicing:
  - Air Force information warfare
  - DoD software development
  - Big 4 consulting experience
  - Fortune 500 to SMB
What is Application Security?

- Ensuring that custom application code performs as expected under the entire range of possible inputs
- Software development focus
  - Traditional InfoSec focuses at the TCP/IP layer
  - Many traditional InfoSec practitioners are ill-equipped to work in the application security space
Cultural Differences

- Traditional InfoSec tends to have a “measure” culture
  - Determine what is in place
  - Audit configurations
- Application development tends to have a “build” culture
  - Create something that did not exist before
  - Project-based deadline environment
  - Certainly the case for custom software development; also largely the case for systems integration work
Why Does AppSec Matter?

- Business critical web apps are Internet-facing
- New laws and regulations govern how data is stored and made available
  - HIPAA
  - Sarbanes Oxley
  - GLB
- 70+% of applications have serious design or coding flaws
  - Studies performed by @Stake and Foundstone
Top 10 Critical Web App Vulnerabilities

- Unvalidated Parameters
- Broken Access Controls
- Broken Account and Session Management
- Cross-site Scripting Flaws
- Buffer Overflows
- Command Injection Flaws
- Error Handling Problems
- Insecure Use of Cryptography
- Remote Administration Flaws
- Web and Application Server Misconfiguration

** The Open Web Application Security Project www.owasp.org
Example Vulnerabilities

- Hidden HTML Field Manipulation
- Cookie Poisoning
- SQL Injection
Hidden HTML Field Manipulation

- Price information is stored in hidden HTML field with assigned $ value
- Assumption: hidden field won’t be edited
- Attacker edits $ value of product in HTML
- Attacker submits altered web page with new “price”
- Still widespread in many web stores
Price Changes via Hidden HTML tags

```
<input name="ComboP" type="hidden" id="ComboP" value=""
$274.85
$2.74
```

(c) Copyright 2005 Denim Group, Ltd.
Price Changes via Hidden HTML tags
Cookie Poisoning

- Browser cookie is used to store user identity information
- Assumption: cookies are set by server side code and not manipulated by users
- Attacker changes cookie and impersonates another user
Cookie Poisoning
Cookie Poisoning

(c) Copyright 2005 Denim Group, Ltd.
Cookie Poisoning
Cookie Poisoning
SQL Injection

- SQL statements are created from a combination of static text and user inputs
- Assumption: users will enter well-formed inputs
- Attacker crafts a custom input to hijack control of the SQL interpreter and execute arbitrary code
An Example: Insecure Design/Code

```java
try {
    string username = request.getParameter("username");
    string password = request.getParameter("password");
    string sSql = "SELECT * FROM User WHERE username = '" + username + "' AND password = '" + password + "'";
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(sSql);
    ...
} catch(Exception ex) {}
Possible Exploit

- Specially crafted input contains SQL control characters
Possible Exploit

- Malicious user sends in a username parameter of: Dcornell'; DROP DATABASE Ecommerce; --

SQL Executed is:
SELECT * FROM User WHERE username = 'Dcornell'; DROP DATABASE Ecommerce; -- AND password = 'whocares'

- Cracker breaks into database and has cleartext access to usernames and passwords, which may be reused on other sites
- Malicious user finds a way to cause an error condition and exploits the unexpected behavior in some manner
An Example: More Secure Design/Code

```java
try {
    string username = request.getParameter(“username”);
    string password = request.getParameter(“password”);
    string passwordHash = MD5.hash(password);
    PreparedStatement stmt = con.prepareStatement(“SELECT * FROM User WHERE username = ? AND passwordHash = ?”);
    stmt.setString(1, username);
    stmt.setString(2, passwordHash);
    ResultSet rs = stmt.executeQuery();
    ...
} catch(SQLException sqlEx) {
    // Actually handle error condition...
}
```
J2EE Specific Examples

- Access Control
  - Authentication
  - Authorization

- Input Validation
  - Stinger framework
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Buffer Overflows
  - Command Injection
Access Control

- This is where security begins
- Responsible for 3 or 4 of the OWASP Top 10
  - Broken Access Controls
  - Broken Account and Session Management
  - Remote Administration Flaws
  - Web and Application Server Misconfiguration (kind of)
- Don’t rely on security through obscurity
- See the application server documentation
J2EE Artifacts

- Setup of users and roles
- Deployment descriptors
  - web.xml
    - Controls access to servlets and JSP pages
  - ejb-jar.xml
    - Controls access to EJBs and EJB methods
    - Very powerful
    - Lots of XML
web.xml for HTTP Basic Auth

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

<login-config>
  <auth-method>BASIC</auth-method>
</login-config>

<security-role>
  <role-name>Administrator</role-name>
</security-role>

Rest of web.xml
```
web.xml for Form Auth

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

<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>login.jsp</form-login-page>
        <form-error-page>login_error.jsp</form-error-page>
    </form-login-config>
</login-config>

<security-role>
    <role-name>Administrator</role-name>
</security-role>

Rest of web.xml

(c) Copyright 2005 Denim Group, Ltd.
More on Form Auth

- Login form:
  - Username: j_username
  - Password: j_password

- See the application server documentation
JAAS

- java.sun.com/products/jaas/
- Java Authentication and Authorization Services
- Standardized API for leveraging or building access controls
  - Most application servers have implementations for JDBC, LDAP
  - Can create your own
- Can be used to secure access to servlets, JSPs, EJB methods, etc
More JAAS

- 3 steps
  - Create a LoginModule
    - Implement javax.security.auth.spi.LoginModule
    - initialize, login, commit, abort, logout methods
  - Create a configuration file
    - $JAVA_HOME/jre/lib/security/java.security
  - Make application JAAS aware
    - Servlets, etc
Input Validation

- “Root of all evil” in application security
- Responsible for 4 of the OWASP Top 10
  - Unvalidated Parameters
  - Cross-site Scripting Flaws
  - Buffer Overflows
  - Command Injection Flaws
J2EE Validation Framework: Stinger

- [http://www.owasp.org/software/validation/stinger.html](http://www.owasp.org/software/validation/stinger.html)

```java
// get the stinger instance so it knows where to find the rules
// validate this request against the rules

Stinger stinger = Stinger.getInstance( this.getServletConfig() );

ErrorList errors = null;
try
{
    errors = stinger.validate( request );
}
catch ( FatalValidationException e )
{
    request.getSession().invalidate();
    out.println( "Invalid HTTP Request" );
    out.close();
    return;
}
```

(c) Copyright 2005 Denim Group, Ltd.
J2EE Cross Site Scripting Defense

- Escape key HTML characters like `< > &
- [Link](http://www-106.ibm.com/developerworks/security/library/s-csscript/)

```java
StringBuffer sbuf = new StringBuffer();

char[] chars = myText.toCharArray();

for (int i = 0; i < chars.length; i++) {
    sbuf.append("&#" + (int) chars[i]);
}
```

(c) Copyright 2005 Denim Group, Ltd.
J2EE SQL Injection Countermeasures

- Use stored procedures:

```java
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");
cs.setInt(1, 75);
cs.setString(2, "Colombian");
ResultSet rs = cs.executeQuery();
```
J2EE SQL Injection Countermeasures

- Use parameterized queries:

```java
PreparedStatement updateSales = con.prepareStatement("UPDATE COFFEE SET SALES = ? WHERE COF_NAME LIKE ?");

updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
```
Buffer Overflows

- Thankfully not typically a big deal in J2EE environments
- JNI code should be wrapped
  - Treat it as if you are crossing a trust boundary

(c) Copyright 2005 Denim Group, Ltd.
Command Injection

- Watch out for System.execute() calls
- Strip out potentially harmful shell control characters: ` `, `;`, `&`, `|`, etc
- Command separators:
  - Windows - `&`
  - UNIX - `;`
- If users are passing in filenames, disallow `'.'` Characters so that `../` cannot be used to “escape” the base directory
- Less of a need to escape these special characters as you might have to do for SQL Injection or XSS situations
  - Not many applications actually accept those characters as legitimate inputs
  - Instead strip them out or disallow sending tainted inputs to the command

(c) Copyright 2005 Denim Group, Ltd.
Resources

- OWASP [www.owasp.org](http://www.owasp.org)
  - WebGoat training tool
  - WebScarab penetration testing tool (proxy)
  - Discussion lists
- @Stake (now Symantec) [www.atstake.com](http://www.atstake.com)
  - netcat TCP/IP tool
  - Whitepapers

(c) Copyright 2005 Denim Group, Ltd.
Questions

Dan Cornell
Denim Group, Ltd.
dan@denimgroup.com
www.denimgroup.com
(210) 572-4400
Responses to Questions

- There were two unanswered questions at the end of the presentation
  - WebScarab and SSL
  - Netegrity and application security
WebScarab and SSL

- As it turns out, WebScarab does support SSL

- [http://www.owasp.org/software/webscarab.html](http://www.owasp.org/software/webscarab.html)

- "Proxy - observes traffic between the browser and the web server. The WebScarab proxy is able to observe both HTTP and encrypted HTTPS traffic, by negotiating an SSL connection between WebScarab and the browser instead of simply connecting the browser to the server and allowing an encrypted stream to pass through it."
Netegrity

- www.netegrity.com
- Purchased by Computer Associates in 2004
- Works with JAAS for authentication and authorization in Java/J2EE environments