J2EE Web Application Security

Introduction to J2EE Web Application Security

J2EE Web Application Security

- **It is a branch of information security that deals with security of web sites and web applications.**
- **J2EE Web Application Security works on the principles of application security, applying them specifically to the web sites and web applications.**

Security threats

Major security threats for Web Application are:

- **Black hat hackers**
  - **Script kiddie**
  - **Hacktivist**
- **Hacking tools**
- **Crackers**
- **Malwares**

Security Standards

- There are many such security threats and they are increasing with the usage of internet. In order to counteract these threats, there was a need for secured systems. There are many different ways to test security flaws and such flaws need to be documented.

Security Risks

- **Web Application Security Risks:**
  - SQL Injection
  - Cross Site Scripting (XSS)
  - Broken Authentication and Session Management
  - Insecure direct object reference
  - Cross site request forgery
  - Security Misconfigurations
  - Insecure Cryptographic Storage
  - Failure to URL Access

Top 15 risks

- Insufficient transport layer protection
- Unvalidated redirects and forwards
- Authentication
- Authorization
- Application Denial Service
- Buffer Overflow
- Path

References

- http://en.wikipedia.org/wiki/Malware

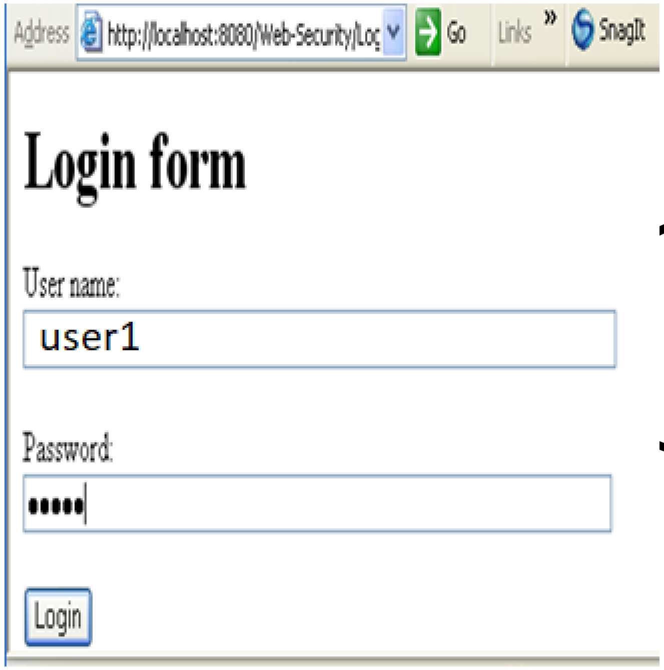
J2EE Web Application Security

SQL Injection

SQL Injection

- **SQL injection occurs when:**
  - Data is entered in our program from an untrusted source.
  - This data is used to dynamically construct a SQL query
- **The main consequences are:**
  - Confidentiality
  - Authentication
  - Authorization
  - Integrity

How does it work



Address http://localhost:8080/Web-Security/Lo Go Links SnagIt

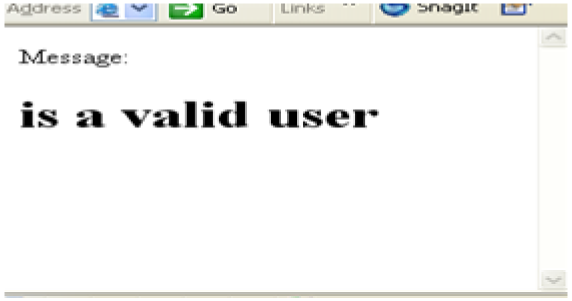
Login form

User name:
user1

Password:
•••••

Login

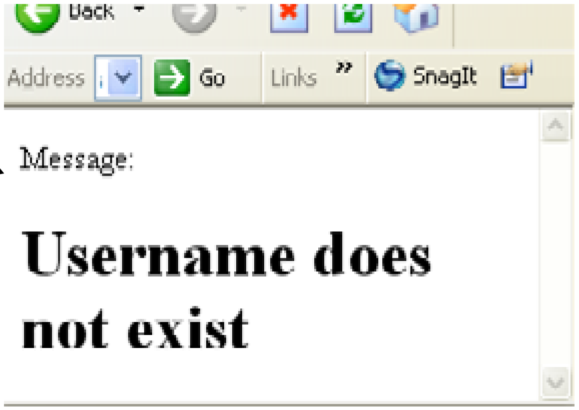
Valid user



Message:

is a valid user

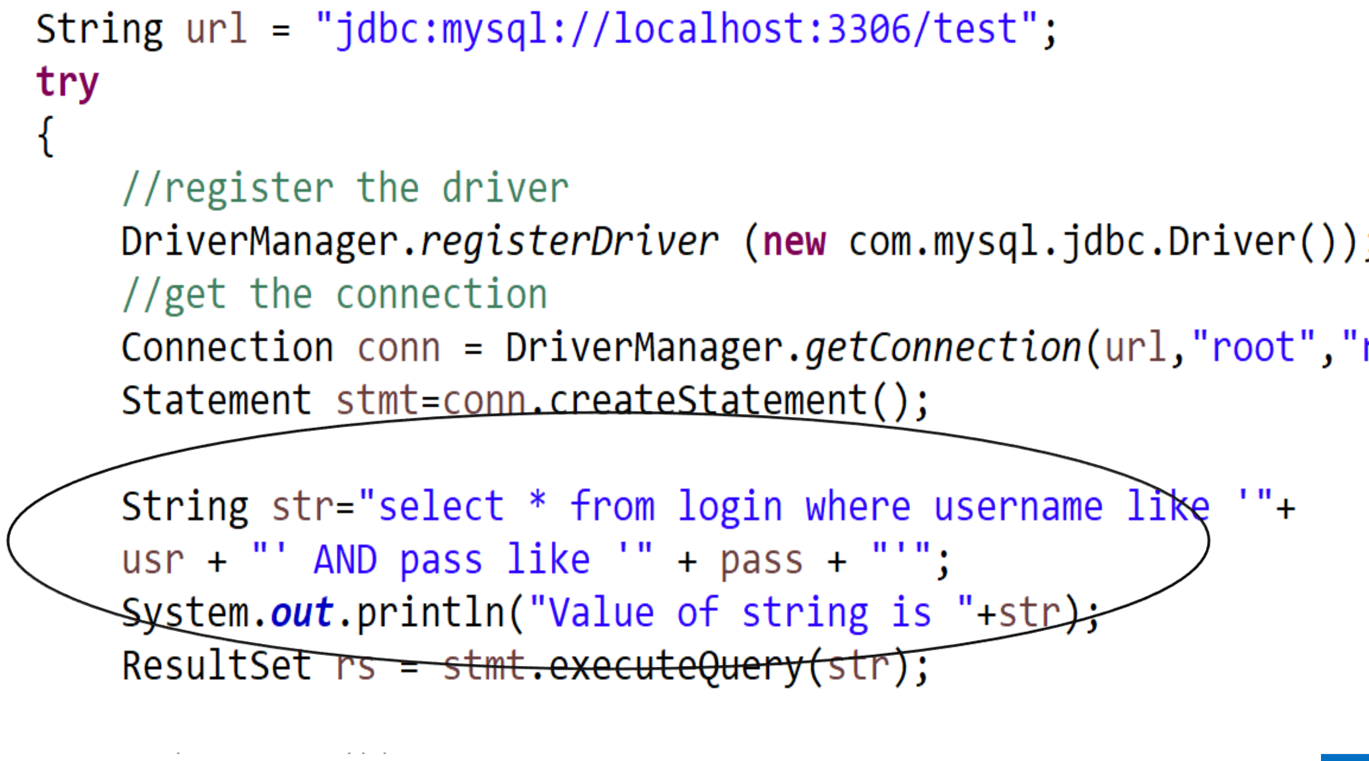
Invalid user



Message:

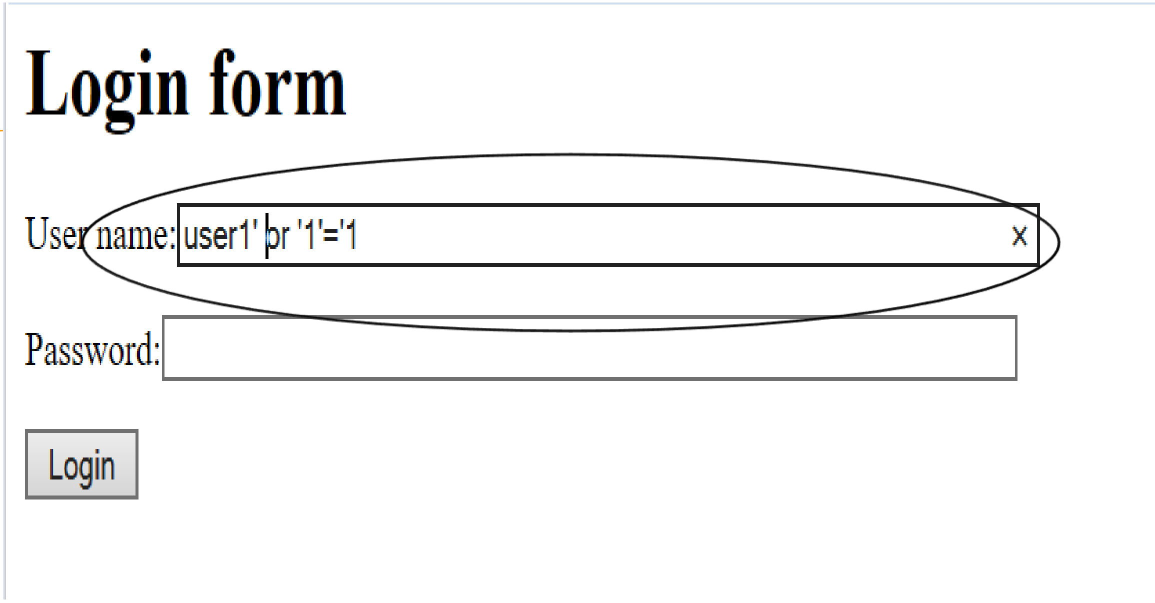
Username does not exist

How does it work



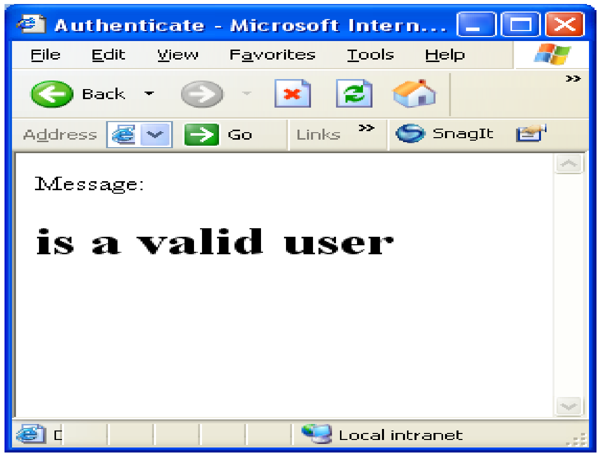
```
String url = "jdbc:mysql://localhost:3306/test";
try
{
    //register the driver
    DriverManager.registerDriver (new com.mysql.jdbc.Driver())
    //get the connection
    Connection conn = DriverManager.getConnection(url,"root","
    Statement stmt=conn.createStatement();

    String str="select * from login where username like '"+
    usr + "' AND pass like '" + pass + "'";
    System.out.println("Value of string is "+str);
    ResultSet rs = stmt.executeQuery(str);
```

How does it work


Login form

User name: user1' or '1'='1

Password:


Authenticate - Microsoft Intern...
File Edit View Favorites Tools Help
Back
Address Go Links SnagIt
Local intranet

Message:

is a valid user

Causes for successful SQL Injection

- **Main causes of successful SQL injection are:**
  - Use of external interpreters
  - Calls to databases not properly validated
  - User input is not filtered for escape characters
  - User supplied field is not checked for type constraints
  - Privilege check not done on each user/role
  - Vulnerabilities can exist within the database server software itself
  - Allowing cross site scripting
  - Logging unnecessary data
  - Absence of stored procedures or PreparedStatement

Measures for handling SQL Injection

- Avoid using external interpreters
- Use prepared statements instead of creating “Statement” at run time.
- Use of stored procedures or prepared statements will help in reducing this risk
- Carefully validate all calls to the databases
- Web application should run with only the minimum privileges
- Perform Input Validation, escape single quotes , reject bad inputs
- Avoid verbose error messages
- Check for type constraints
- Avoid logging data that came from external inputs

References

- http://en.wikipedia.org/wiki/SQL_injection

J2EE Web Application Security

Cross Site Scripting

What is Cross Site Scripting

- **Cross-site scripting (XSS) enables attacker to inject client-side script into web pages viewed by other users.**
  - The attacker can post a page to the user that might look like Trusted site, but in reality is the attacker's site. The attacker may have provided a link to his site with an embedded script.
  - Such embedded script can be executed on client machine.
  - As it crosses sites, it is called as Cross Site Scripting.

How does it work – Scenario 1

- **This could be some phishing website, looking similar to mybank.com, but not authentic mybank.com, user may not check address bar**

Causes for successful XSS

- Poor validations
- Origin of a script not checked before modifying it or using it in program
- Allowing any script program to run, without prompting end user with a warning
- Allowing input to form a script for opening a file or executing a system call
- Use of SSI

Measures for handling XSS

- **Programmers need to:**
  - Carefully validate all data before processing it
  - Check origin of a script not checked before modifying or using it in program
  - Write scripts with programming languages like PERL
  - Use ‘server-side includes’ minimal or not at all
  - Restrict or disable external program execution in your Web server
  - Restrict ability of the web server to execute external programs at the OS level
  - Use HttpOnly flag if possible
- **End users need to check authenticity of script before allowing it to execute on machine**

References

- http://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.watchguard.com/infocenter/editorial/135142.asp
- http://www.mozilla.org/projects/security/components/jssec.html#writing

J2EE Web Application Security

Broken Authentication and Session Management

Introduction

- Broken authentication and session management is a type of vulnerability, where some aspect of handling user authentication and managing active sessions is failed

Causes

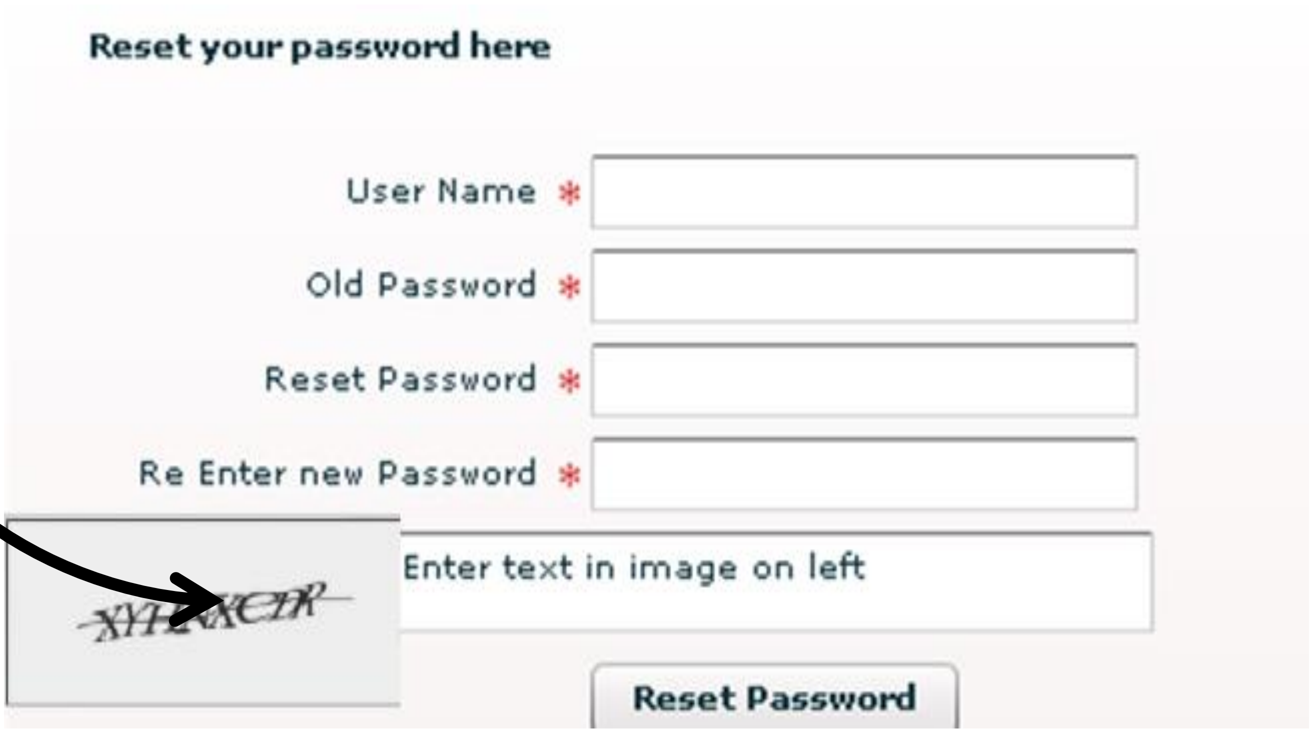
- Credentials are transferred to the server in insecure manner
- Session invalidation is not managed properly
- Custom session management mechanism could be complex but, not very reliable
- Using URL Rewriting for session management
- Not implementing a strong authentication mechanism
- Not Implementing a strong password policy

Mechanisms for protection

- **Use SSL/TLS at least for authentication related processes**
- **Use of inbuilt session management mechanism**
- **Do not accept any session identifier from the URL or in the request**
- **Limit code of custom cookies for authentication or session management**
- **Implement a strong authentication mechanism, use Captcha while resetting the password**
- **Implement a strong password policy when allowing passwords**
- **Ensure that every page has a logout link, also keep timeout period for session**

Measure: Use of Captcha

Captcha


Reset your password here

User Name *

Old Password *

Reset Password *

Re Enter new Password *

Enter text in image on left

References

- http://misc-security.com/blog/2009/08/broken- authentication-and-session-management/

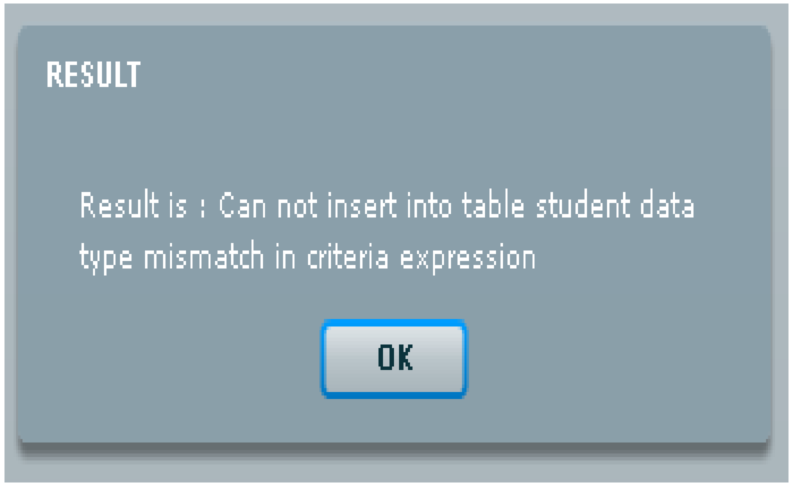
J2EE Web Application Security

Insecure direct object reference

Insecure Direct Object Reference

- **A direct object reference occurs when an internal implementation like file, directory, database table, parameter or url is exposed.**
- **Such exposed objects are susceptible to attack.**

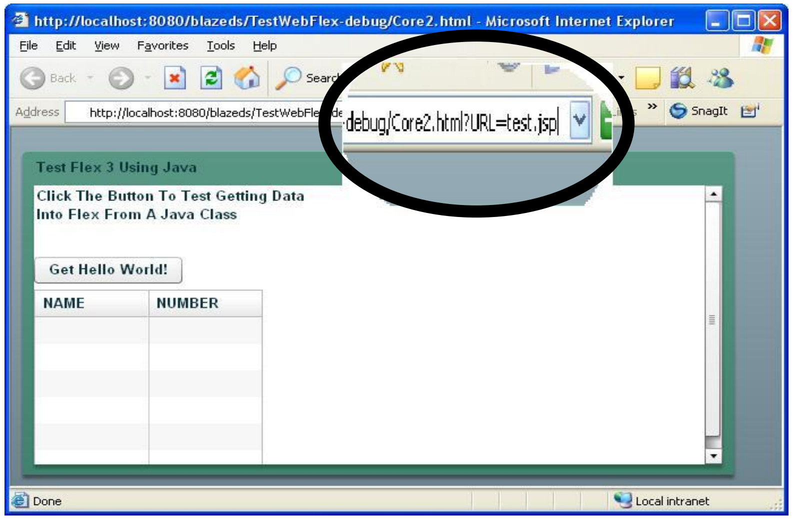
How does it work

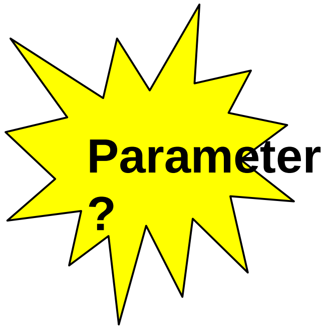

RESULT

Result is : Can not insert into table student data type mismatch in criteria expression

Table reference ?

How does it work


http://localhost:8080/blazeds/TestWebFlex-debug/Core2.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Address
http://localhost:8080/blazeds/TestWebFle
debug/Core2.html?URL=test.jsp
SnagIt
Test Flex 3 Using Java
Click The Button To Test Getting Data Into Flex From A Java Class
Get Hello World!
NAME
NUMBER
Done
Local intranet



Causes

- Private object references like filenames, parameters are exposed to end users
- Absence of code to validate private object references
- Objects are not authenticated properly
- URL or parameters containing patterns like ../
- Exposing direct references to database objects

Mechanisms for Protection

- **Programmer needs to:**
  - Avoid exposing private object references to end users
  - Write a code to validate any private object references before using them
  - For all referenced objects, verify authentication
  - Never expose direct references to database objects
  - Code review

References

- **http://serdarbuyuktemiz.blogspot.com/2008/09/insecure-direct-object-reference.html**
- **http://www.12robots.com/index.cfm/2010/1/19/Insecure-Direct-Object-Reference--Security-Series-15**

J2EE Web Application Security

Cross Site Request Forgery

Cross Site Request Forgery(CSRF)

- It is a type of vulnerability in which unauthorized commands are transmitted from a trusted user
- It is also called as one-click attack or session riding

Characteristics of CSRF

- **It involves sites relying on a users identity**
- **CSRF exploits trust that a site has in the users browser**
- **It requires some trick that makes the user browser send HTTP requests to a target site**
- **It works on sites that do not check the “referrer” header or a client with a browser that allows referrer spoofing**
- **The attacker must find a form submission, right values for all the form's components, parameters**
- **Attacker needs to convince/distract the victim to open target site**

Causes for successful CSRF

- Users identity from cookies
- Referrer spoofing
- Clients not logging out
- Clients not alert

Measures for handling CSRF

- **Programmers need to:**
  - Write a code to check the HTTP “referer” header.
  - Limit session lifetime and invalidate session as user logs off.
  - Should have some user specific information that should be required for critical processes like transfer of funds. For example, some banks request for transaction password, or enter some number which is on the debit card of an user.

Measures for handling CSRF

- **Client needs to:**
  - Use a browser that prevents referrer spoofing
  - Disable cookies
  - Log out when transactions are done
  - Be careful while chatting with unknown people

–

References

- http://en.wikipedia.org/wiki/ Cross-site_request_forgery

J2EE Web Application Security

Security Misconfigurations

Security Misconfiguration

- **Application Misconfiguration attack exploits configuration weaknesses in the web application. It can lead to unauthorized access to sensitive information that can adversely affect privacy of an individual, business secrets or even the security.**
- **Generally installations are done using default settings, which generally have applications with many unnecessary features. For example, debugging, which gives extra information that can help hacker to gain access to the sensitive information even without having proper authentication.**
- **Default installations may also have well-known usernames and passwords like "sa/ " or "scott/tiger", that can help hacker to gain access to the database.**
- **If proper file access permissions are not given, then hacker can get access to the sensitive information.**

Causes for Security Misconfiguration

- **Exceptions not handled properly**
  - If we use printStackTrace or getMessage methods in catch block, then in case of an exception, hacker will come to know about what code is used for processing the request, as exception will cause actual error display on the screen.

- **File access not controlled by access permissions**
- **Data not transmitted in secure manner over the network**
  - Use HTTPS protocol for sensitive information
- **Not Using the “Secure” flag**

  -
    ```
    <session-config>
       <cookie-config>
                       <secure>true</secure>
        </cookie-config>
    </session-config>
    ```

- **Not Using the “HttpOnly” flag**
  - Cookies cannot be accesed by client side script
- **Use of URL parameters for session tracking**
- **Not setting a session timeout period**

Causes for security misconfiguration

- **Not Using the HttpOnly Flag**

```
<session-config>
    <cookie-config>
      <http-only>true</http-only>
    </cookie-config>
  </session-config>
```

If cookies are created using "HttpOnly" flag, it cannot be accessed using client side scripts.

- **Using URL Parameters for Session Tracking**

```
<session-config>
    <tracking-mode>COOKIE</tracking-mode>
  </session-config>
```

Is available in Servlet 3.0 API’s Session id as URL parameter is generally stored in many locations like browser history, logs, referrer log files etc.

- **Not Setting a Session Timeout**

```
<session-config>
    <session-timeout>31</session-timeout>
  </session-config>
```

If such setting is done, then session will automatically expire after 31inactive minutes.

Mechanism for protection

- **Handling all exceptions/errors using proper using error pages.**
- **Managing file access by specifying access permissions**
- **Transmitting data in secure manner over the network**
- **Using the “Secure” flag**
- **Using the “HttpOnly” flag**
- **Not using URL parameters for session tracking**
- **Setting session timeout period for invalidating session**

Mechanism for protection

- **Managing file access by specifying access permissions**

In web.xml we can configure for it using:

```
<security-constraint>
            <web-resource-collection>
                        <web-resource-name>MyProject</web-resource-name>
                         <url-pattern>/Report*</url-pattern>
                         <http-method>GET</http-method>
             </web-resource-collection>
             <auth-constraint>
                         <role-name>user</role-name>
              </auth-constraint>
  </security-constraint>
```

- We can create various users and roles by using application servers like JBoss. Further, we can specify access rights for each user/role over the application.
- In the example above, role named as “user” has access to all files with name starting with “Report” but can invoke only GET method within those resources.

References

- http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files/
- http://projects.webappsec.org/w/page/13246914/Application-Misconfiguration

J2EE Web Application Security

Insecure Cryptographic Storage

Insecure Cryptographic Storage

- Some applications do not use encryption mechanism, but use poor cryptography.
- It either uses inappropriate ciphers or does not use strong ciphers properly.
- This can lead to disclosure of sensitive / secret /personal information.

Causes

- **Use of user defined/custom cryptographic algorithms**
- **Use of weak algorithms like MD5 or SHA1**
- **Generating keys online**
- **Storage of private keys on machines available online**
- **Use of weak encoding, which can be easily decoded**

Mechanism for Protection

- **Do not use user defined cryptographic algorithms or weak algorithms**
- **Do not generate keys online**
- **Use Strong encryption**

References

- http://serdarbuyuktemiz.blogspot.com/2008/09/insecure -cryptographic-storage.html

J2EE Web Application Security

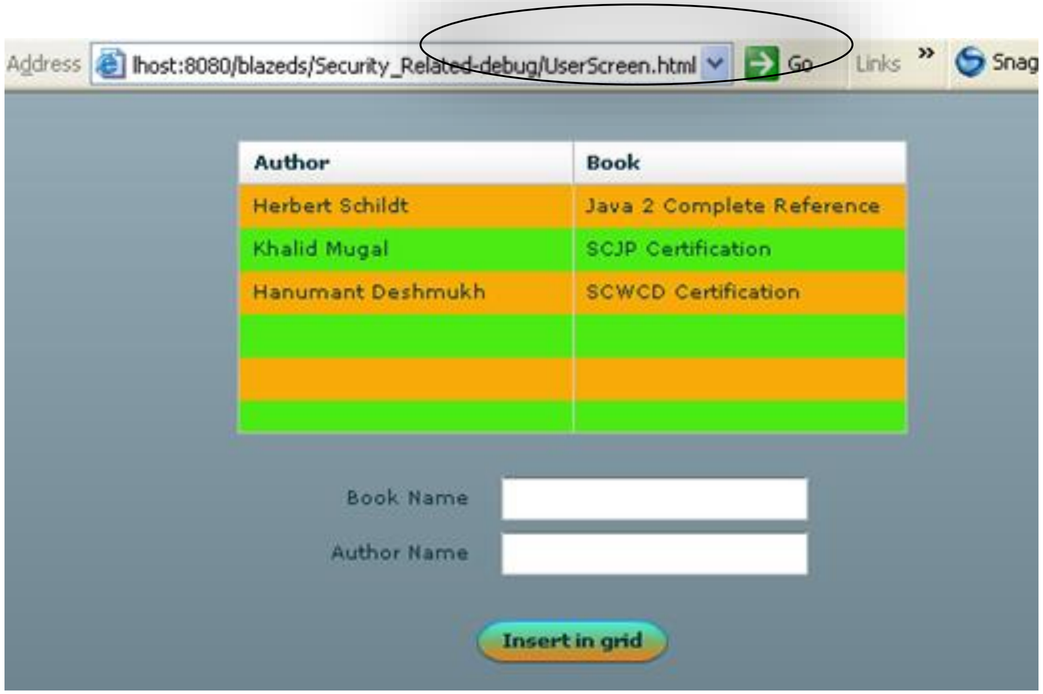
Failure to URL Access

What is Failure to URL Access?

- **Forced browsing: It is the primary attack method for failure to URL access.**
- **It requires guessing the links and brute force techniques to find unprotected pages.**

Examples

- "Hidden" or "special" URLs



Address lhost:8080/blazeds/Security_Related-debug/UserScreen.html

Author	Book
Herbert Schildt	Java 2 Complete Reference
Khalid Mugal	SCJP Certification
Hanumant Deshmukh	SCWCD Certification

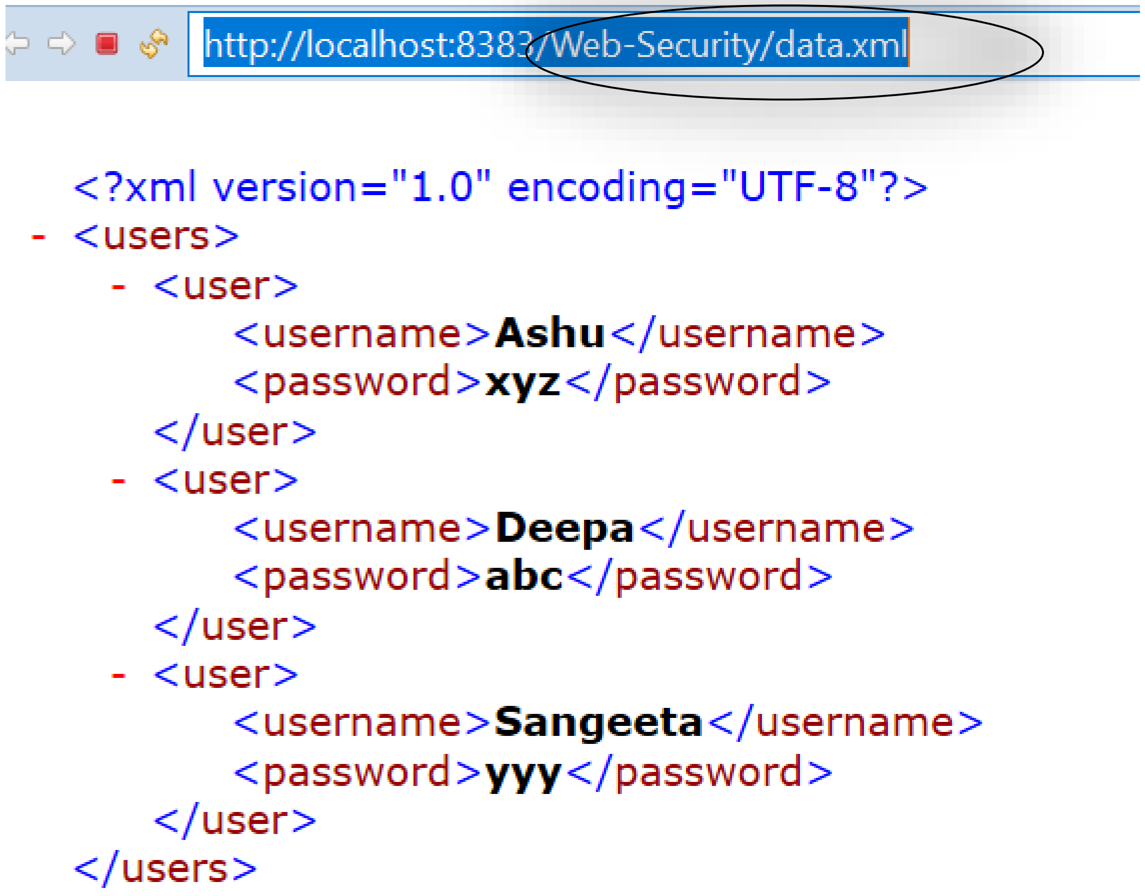
Book Name

Author Name

Loophole:
If you have UserScreen.html, You may have XXXRoleScreen.html as well, let me check it

Examples Contd…

- **Access to "hidden" files**

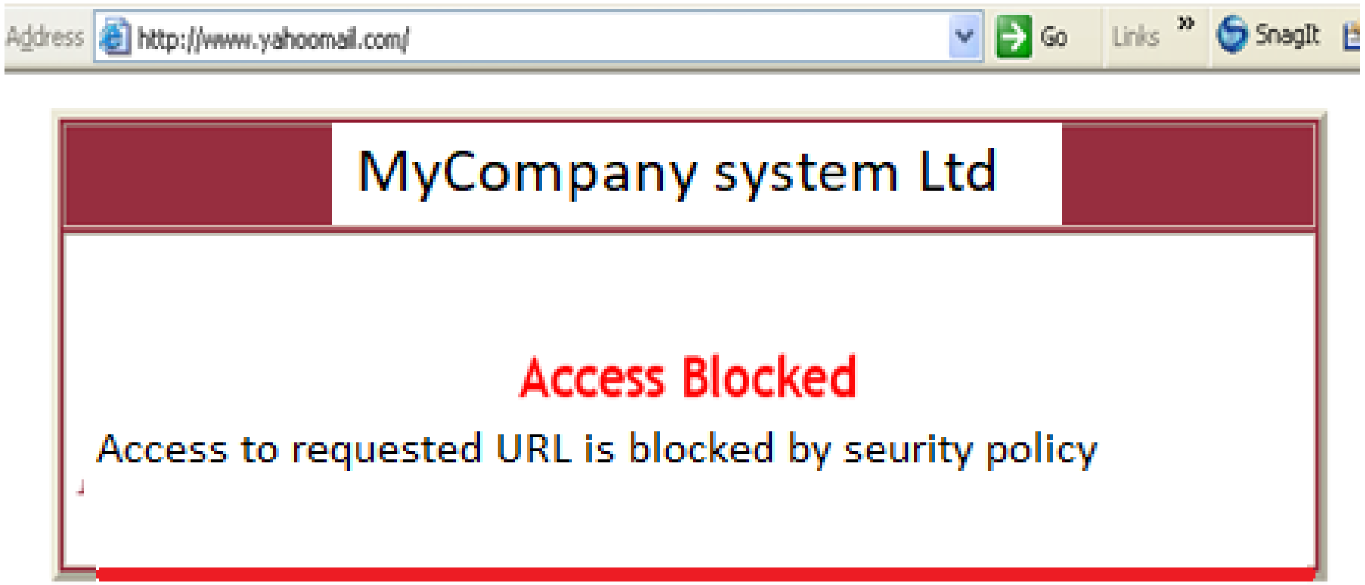


http://localhost:8383/Web-Security/data.xml

```
  <?xml version="1.0" encoding="UTF-8"?>
- <users>
  - <user>
        <username>Ashu</username>
        <password>xyz</password>
    </user>
  - <user>
        <username>Deepa</username>
        <password>abc</password>
    </user>
  - <user>
        <username>Sangeeta</username>
        <password>yyy</password>
    </user>
  </users>
```

Examples Contd…

- Out of date or insufficient security



MyCompany system Ltd

Access Blocked

Access to requested URL is blocked by seurity policy

Verifying Security

- **Automated Approach**
  - Various tools can be used to find it
  - But it is generalized
- **Manual Approach**
  - Better use
  - Can be specific to your application

Verifying Security Contd…

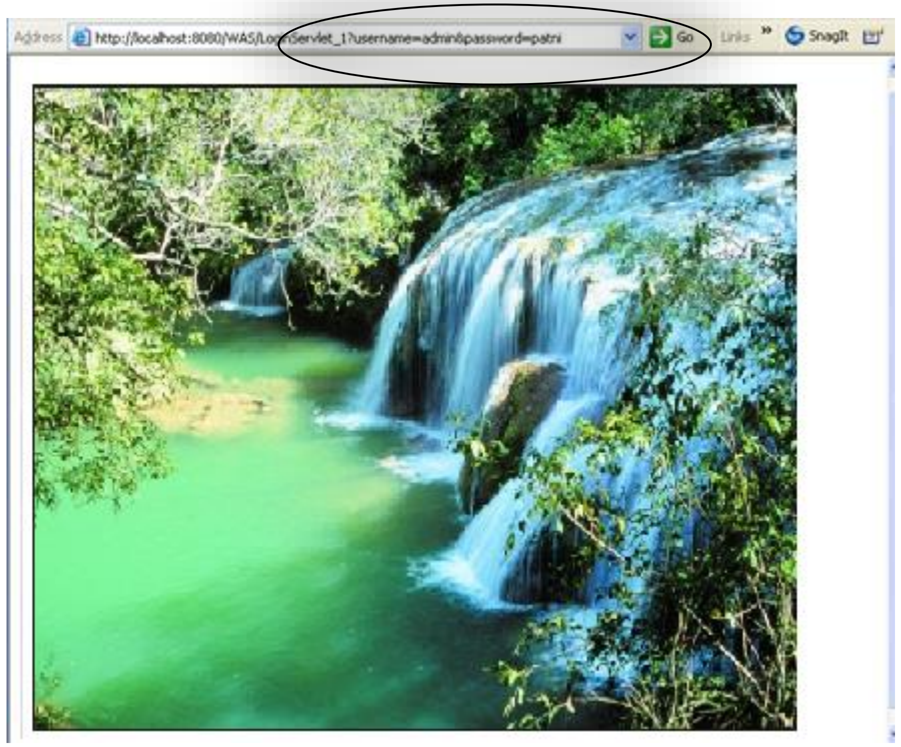
- Using doGet method

Concern:

Parameters NAMES are clearly seen on the address bar

Verifying Security Contd…

On valid login, it displays the following screen:



http://localhost:8080/WAS/LoginServlet_1?username=admin&password=patni

Verifying Security Contd…

- No access control provided
- Loophole:
  - No need to go through THE site AUTHENTICATION, I can directly access images; that’s what I wanted ☺

Protection

- **Creating a matrix to map the roles and functions:**

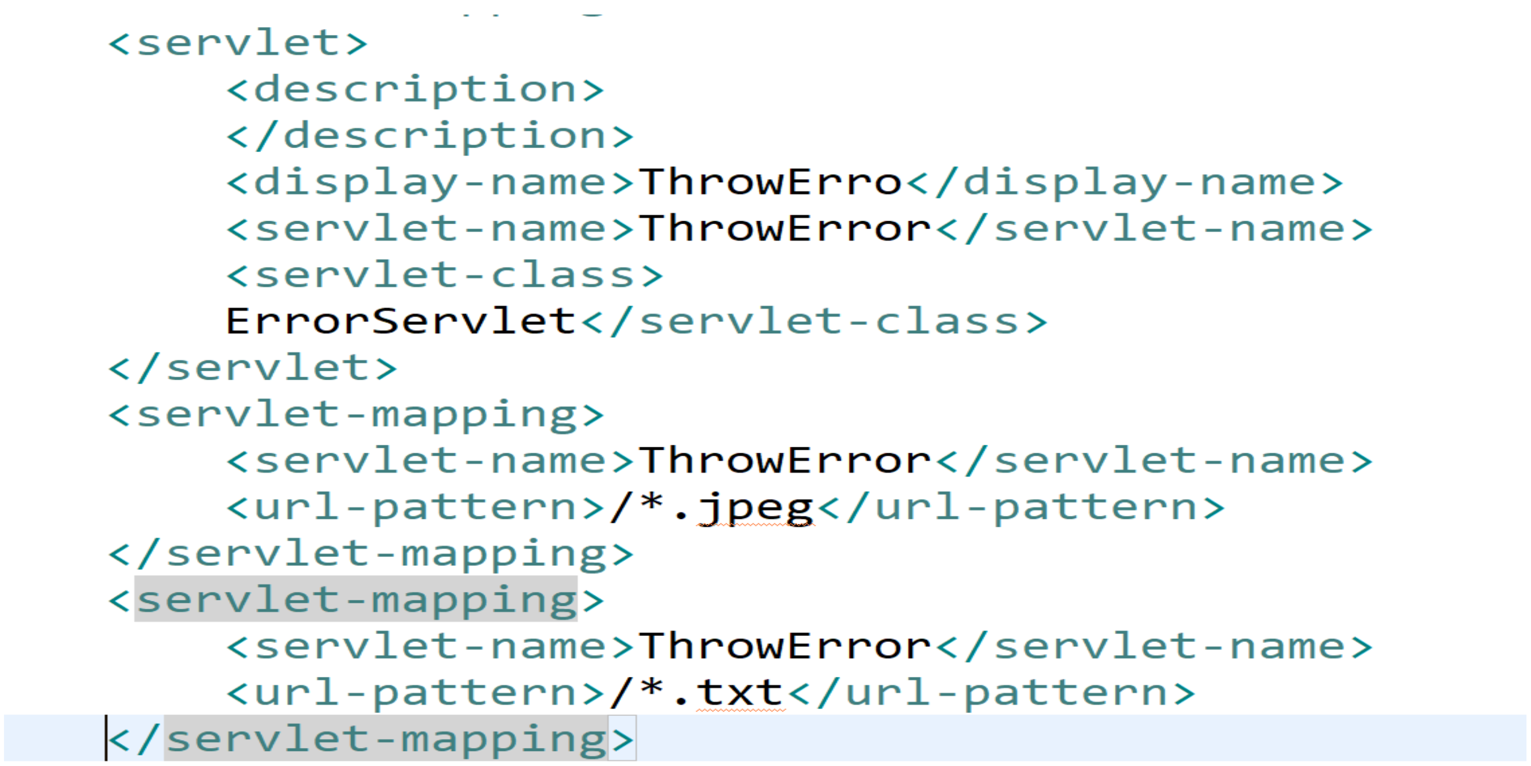
Roles/Functions	Request to change own password	Approve leaves	Change status of installation request
Employee	√		
Manager	√	√	
Network engineer	√		√

Protection Contd…

- **For protecting URL, we need to:**
  - Ensure that the access control matrix is part of the business, architecture, design and presentation of the application
  - Ensure that all the URLs and business functions have an effective access control mechanism
  - Perform a penetration testing
  - Block access to all file types that your application should not directly provide to the end user

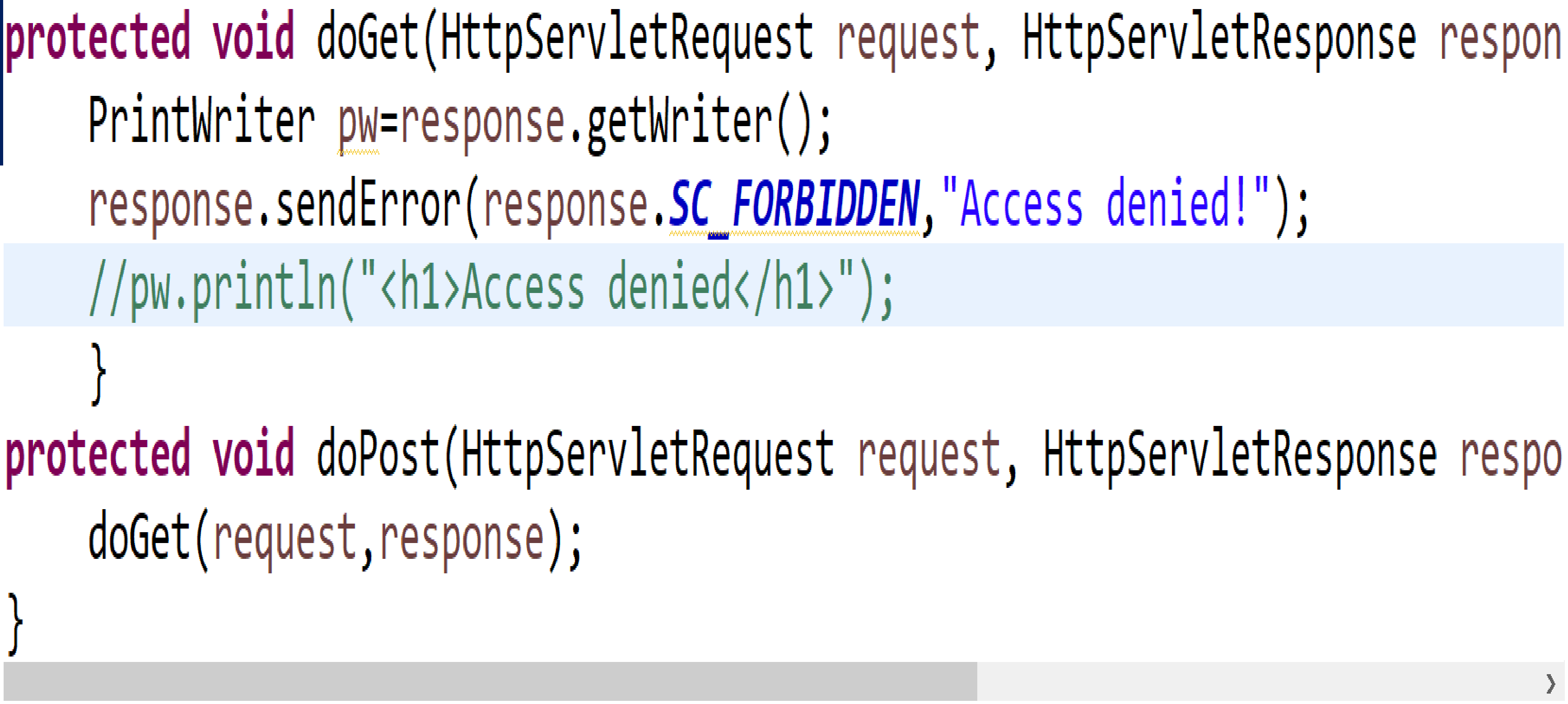
Restricting access

Add following restrictions in web .xml



```
<servlet>
    <description>
    </description>
    <display-name>ThrowErro</display-name>
    <servlet-name>ThrowError</servlet-name>
    <servlet-class>
    ErrorServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ThrowError</servlet-name>
    <url-pattern>/*.jpeg</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>ThrowError</servlet-name>
    <url-pattern>/*.txt</url-pattern>
</servlet-mapping>
```

Servlet code



```
protected void doGet(HttpServletRequest request, HttpServletResponse respon
    PrintWriter pw=response.getWriter();
    response.sendError(response.SC_FORBIDDEN,"Access denied!");
    //pw.println("<h1>Access denied</h1>");
    }
protected void doPost(HttpServletRequest request, HttpServletResponse respo
    doGet(request,response);
}
```

Checklist

- **We should follow the following checklist:**
  - Restrict access to target authenticated users
  - Role based permissions on each URL should be used
  - Disallow requests to page types like log files, source files, images, etc.
  - Each URL in your application should be protected by:
    - External filter, like Java EE web.xml.
    - Internal checks in your code should be done.

J2EE Web Application Security

Insufficient Transport Layer Protection

Access to System

- **Who can access my system**
  - Vendors
  - Clients
  - External hackers
  - Employees
  - Internal hackers

Advantages of Transport Layer Protection

- Protection of web application data from unauthorized disclosure and modification while data is getting transfered
- It provides authentication of the server to the client. Also, it can do the client authentication to the server.
- It provides integrity guarantee and replay prevention
  - Data transfer in secure manner is guaranteed

Mechanisms for Transport Layer Protection

- Use TLS
- Transmit only signed messages
- Use strong algorithms for encryption/decryption
- Manage keys/certificates properly
- Verify SSL certificates before using them
- Use tested mechanisms when sufficient
- e.g., SSL

PKI (Public key Infrastructure)

- **Certification authority(CA)**
  - These are registered authorities

- **Verifying Authority(VA)**
- **Registering Authority(RA)**

Configuring SSL Aware Security Domain

- **Tomcat Web Server has built in support for SSL**
- **Authenticate clients based on certificates, then define a SSL aware Security Domain.**

Configuration in Server.Xml

- **Code Snippet:**

```
<Connector port="8443" protocol="HTTP/1.1"
SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
              keystoreFile="c:\mykeystore\mykeystr.txt"
              keystorePass="root123" />
```

Configuration in Web.Xml Contd…

- **Code Snippet:**

```
<security-constraint>
<web-resource-collection>
<web-resource-name>securedapp</web-resource-name>
<url-pattern>/*</url-pattern>
<url-pattern>/shoppingcart/*</url-pattern>
<http-method>POST</http-method>
</web-resource-collection>
<user-data-constraint>
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

References

- http://projects.webappsec.org/w/page/13246945/Insufficient-Transport-Layer-Protection
- http://docs.jboss.org/jbossas/jboss4guide/r5/html/ch8.chapter.html

J2EE Web Application Security

Unvalidated Redirects and Forwards

Unvalidated Redirects and Forwards

- **Web applications generally use lots of redirects and forwards.**
- **Unvalidated redirects and forwards happen when, attacker redirects user to unintended pages or websites like phishing, malware sites.**
- **In absence of proper validation, attackers can use unvalidated forwards to bypass authentication or authorization checks.**

Causes for Unvalidated Redirects and Forwards

- Security breach
- Source of request not checked before forwarding or redirecting
- User accesses link in email or forum, or from social networking site

Mechanism for Protection

- **Multi staged security**
- **Check source of request**
- **User should be alert while accessing links**
- **Use “referer” header**

References

- http://www.infosecurity-us.com/view/8930/owasp-updates-application-vulnerability-list/
- http://www.imperva.com/docs/TB_SecureSphere_OWASP_2010-Top-Ten.pdf

J2EE Web Application Security

Authentication Vulnerability

What is Authentication Vulnerability?

- Authentication vulnerability occurs when the unauthenticated person gets access to the Web application.

Causes for Authentication Vulnerability

- **Main causes for authentication vulnerability are:**
  - Hard-coded password
  - Disallowing password aging
  - Empty String password
  - Authentication bypass via assumed immutable data
  - Failure to drop privileges when reasonable
  - Unsafe mobile code
  - Unsecure password storage

Mechanisms for Protection from Authentication Vulnerability

- **Do not hard-code password**
- **Allow password aging**
- **Use non-empty String password**
- **Do not assume immutable data for authentication**
- **Drop privileges when reasonable**
- **Use safe mobile code**
- **Use secure password storage**

J2EE Web Application Security

Lesson 13: Authorization Vulnerability

What is Authorization Vulnerability?

- **Authorization vulnerability is exposed when an user gets access to the unauthorized page, which is not meant for him/ her.**
- **For example, employee having “programmer” role accessing links meant for “Manager” role, which can approve leaves requested by “Programmer”.**

Causes for Authorization Vulnerability

- **Main causes for authentication vulnerability are:**
  - Access control enforced by presentation layer only
  - File Access Race Condition: TOCTOU
  - Least Privilege Violation
  - Privilege Management misused
  - Using “referer” field for authorization

Mechanisms for Protection from Authorization Vulnerability

- Access control enforced at many layers.
- Measure for File Access Race Condition:
  TOCTOU(Time of check and Time of Use)
- Assigning least privilege
- Strong Privilege Management
- Not relying on “referer” field for authorization

References

- https://www.owasp.org/index.php/b
- http://www.usenix.org/event/fast08/tech/full_papers/tsafrir/tsafrir_html/

J2EE Web Application Security

Application Denial of Service

What is Application Denial of Service?

- Denial of Service (DoS) is an attempt to make a computer resource unavailable to its intended users.
- Most common method of this type of attack involves saturating the target machine with large number of requests, such that it cannot respond to legitimate traffic, or responds very slowly as if it is unavailable.

Causes for DoS

- **Main causes for DoS are:**
  - Absence of firewalls
  - Use of Switches or Routers without ACL (Access Control List) capabilities
  - Absence of IPS (Intrusion Prevention System)
  - Lack of proactive testing
  - Absence of blackholing, sinkholing and clean pipes

Mechanisms for Protection from Authorization Vulnerability

- **Use of firewalls**
- **Use of Switches or Routers having ACL (Access Control List) capabilities**
- **Use of IPS (Intrusion Prevention System)**
- **Proactive testing**
- **Using of blackholing, sinkholing and clean pipes**

References

- http://en.wikipedia.org/wiki/Denial-of-service_attack
- http://en.wikipedia.org/wiki/Denial-of-service_attack#Performing_DoS-attacks
- http://www.askstudent.com/security/sinkholes-in-network-security-5-easy-steps-to-deploy-a-darknet/
- http://www.nexusguard.com

J2EE Web Application Security

Buffer Overflow

What is Buffer Overflow Vulnerability?

- **Buffer contains a finite amount of data as it has size limitation.**
- **When a program or process tries to store more data than the buffer size, it results in buffer overflow.**
- **Since the extra data needs to be stored somewhere, it may result into overflow to adjacent buffers, corrupting or overwriting data held in them.**

Causes for Buffer Overflow Vulnerability

- **Main causes for Buffer Overflow vulnerability are:**
  - Insecure code
  - Improper validation

Mechanisms for Protection from Buffer Overflow Vulnerability

- **Secure code**
- **Proper Validation**

References

- http://searchsecurity.techtarget.com/definition/buffer-overflow
- http://en.wikipedia.org/wiki/Buffer_overflow#Stack-based_exploitation

J2EE Web Application Security

Path Vulnerability

What Is Path Vulnerability?

- Attackers can access files that are not intended to be accessed. Generally it occurs due to invalidated user input while dynamically creating a file path.

Causes for Path Vulnerability

- **Main causes for Path Vulnerability are:**
  - Path Traversal Attack
    - Relative Path Traversal Attack
    - Absolute Path Traversal Attack
  - Path Equivalence Attack

Mechanisms for Protection from Path Vulnerability

- Secure Path Traversal
- Not allowing use of Path Equivalence

References

- http://cwe.mitre.org/data/published/cwe_v1.3.pdf