

P. 1653



SESSION FIXATION AND ITS PREVENTION IN WEB-BASED APPLICATIONS

A PROJECT REPORT

Submitted by

RANJANI.P	71202205036
SIVARANJANI.V.V	71202205048

in partial fulfillment for the award of the degree

of

**BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY**

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

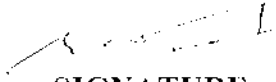
ANNA UNIVERSITY; CHENNAI 600 025

MAY 2006

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “SESSION FIXATION AND ITS PREVENTION IN WEB-BASED APPLICATIONS” is the bonafide work of “**RANJANI.P, SIVARANJANI.V.V**” who carried out the project work under my supervision.



SIGNATURE

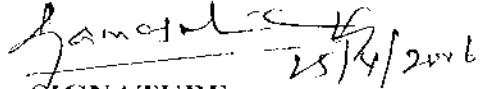
Dr.G.Gopalsamy

HEAD OF THE DEPARTMENT

Information Technology

Kumaraguru College of Technology

Coimbatore-641 006



SIGNATURE

Mr.K.Ramasubramanian

SUPERVISOR

Lecturer

Information Technology

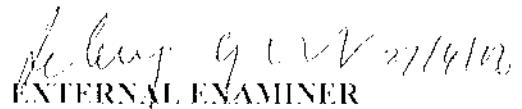
Kumaraguru College of Technology

Coimbatore-641 006

The candidates with University Register Nos. 71202205036, 71202205048 were examined by us in the project viva-voce examination held on 2.1.16.06.



INTERNAL EXAMINER



EXTERNAL EXAMINER

ACKNOWLEDGEMENT

With all humility and submissiveness we surrender myself at the 'Divine Feet' of God and submit our foremost gratitude for having graciously blessed us with knowledge, skill and enthusiasm.

We express our sincere thanks to **Dr.K.Arumugam, B.E (Hons), M.S.(U.S.A), MIE.,** Correspondent, Kumaraguru College of Technology and the management, for providing us an opportunity to undertake this project work.

We express my sincere gratitude to **Dr. K.K.Padmanabhan,** esteemed Principal, Kumaraguru College of Technology, Coimbatore, for permitting us to undergo a project work.

We express sincere thanks to **Dr.G.Gopalsamy, Ph.D.,** Head of the Department, Department of Information Technology for the immense support he has provided us throughout our project.

We deep gratitude and thanks are due in no small measure to our project coordinator **Prof.K.R.Baskaran, B.E, M.S ,** Department of Information Technology, for his constant support, encouragement and vivacious guidance.

We would like to express our heartfelt thanks to our guide, **Mr.K.Ramasubramanian, M.C.A, M.E.,** Lecturer, Department of Information Technology, for his everlasting counseling and untiring help throughout our project.

We thank our beloved parents, friends who have been a pillar of support in all our endeavors, and the department teaching and non teaching staff for their support in completing the project.

DECLARATION

We,

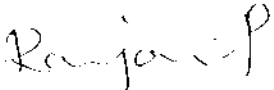
Ranjani.P 71202205036

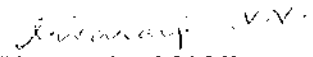
Sivaranjani.V.V 71202205048

Declare that the project entitled **“SESSION FIXATION AND ITS PREVENTION IN WEB-BASED APPLICATIONS”**, submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) Degree, is a record of original work done by us under the supervision and guidance of Mr.K.Ramasubramanian, Lecturer, Department of Information Technology, Kumaraguru College of Technology, Coimbatore.

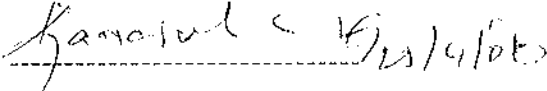
Place: Coimbatore

Date: 27.4.2016


[Ranjani.P]


[Sivaranjani.V.V]

Project Guided by



[Mr.K.Ramasubramanian, M.C.A, M.E]

Lecturer.

ABSTRACT

The project entitled **“PREVENTION OF SESSION FIXATION IN WEB-BASED APPLICATIONS”** deals with implementation of various kinds of security in session management. The main objective is to provide security to web users.

The basic idea behind session management is that the server generates a session identifier at some early point in user interaction, sends this identifier to the user’s browser and make sure that the same identifier will be sent back by the browser along with each subsequent request without any intruder’s invasion.

The session identifiers become identification tokens for the users, and servers can use them to maintain session data, creating a session-like experience to the users. A web application authenticates a user without first invalidating the existing session, thereby continuing to use the session already associated with the user. An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session.

The web application generates a unique session identifier to the user only if he is successfully authenticated. This identifier is bound to the browser’s network address and SSL certificate for every transaction. This enhances security for every request and response.

Session destruction takes place on both server and browser either by logging out or timeout. Thus the user is able to have secure access across the web.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF TABLES	ix
	LIST OF ABBREVIATIONS	x
1.	INTRODUCTION	1
	1.1 The existing system and its limitations	1
	1.2 The proposed system and its advantages	3
2.	SYSTEM REQUIREMENTS ANALYSIS	4
	2.1 Project Definition	4
	2.2 Project plan	4
	2.3 Software Requirements Specification	6
	2.3.1 Purpose	6
	2.3.2 Scope	6
	2.3.3 Definitions	6
	2.3.4 Abbrevations	7
	2.3.5 General Description	7
	2.3.6 Specific Requirements	8
3.	SYSTEM STUDY	10
	3.1 Tomeat Server	10
	3.1.1 Introduction to tomeat server	10
	3.1.2 Installation of Tomeat Server	12

	3.1.3 Deployment	15
	3.2 Java Server Pages	17
4.	DESIGN DOCUMENTS	19
	4.1 Introduction	19
	4.2 Database Design	19
	4.2 Process Design	20
5.	PRODUCT TESTING	23
	5.1 Unit Testing	23
	5.2 Validation Testing	23
	5.3 Output Testing	24
	5.4 Integration Testing	24
	5.5 System Testing	24
6.	FUTURE ENHANCEMENT	25
7.	CONCLUSION	26
8.	APPENDIX	27
	8.1 Sample Code	27
	8.2 Screenshots	50
9.	REFERENCES	55

LIST OF FIGURES

Figure No.	Name	Page No.
1.	Tomcat Class Loaders	10
2.	Three Steps of Session Fixation	20

LIST OF TABLES

Table No.	Name	Page No.
1.	User Details Table	19

LIST OF ABBREVIATIONS

1. SID - Session Identifier
2. SSL - Secure Socket Layer
3. URL - Uniform Resource Locator
4. JSP - Java Server Pages
5. HTTP - Hyper Text Transfer Protocol
6. IP - Internet Protocol
7. JDK - Java Development Kit

1. INTRODUCTION

HTTP(Hyper Text Transfer Protocol) is a stateless protocol, which means that it provides no integrated way for a web server to maintain states throughout user's subsequent requests. To overcome this problem, web applications implement various kinds of session management. Very often, session identifiers are not only identification tokens, but also authenticators. This means that upon login, users are authenticated based on their credentials (username/password, digital certificates) and issued session identifiers that will effectively serve as temporary static passwords for accessing their sessions. Web session security is basically focused on preventing various types of attacks against session identifiers.

1.1 EXISTING SYSTEM AND ITS LIMITATIONS

Almost all of today's stateful web-based applications use session identifiers to associate a group of online actions to a specific user. This has security implications because many state mechanisms that uses session identifiers also serve as authorization and authentication mechanisms-purposes for which they were not well designed. Session identifiers are usually long alphanumeric strings transmitted between client and server either within cookies or directly within URIs(Uniform Resource Locator).Once a user has logged on into an application the session ID's can server as stored authentication mechanisms so that the user does not have to retype a password after each click within the website.

Session identifiers are stored in a cookie held by the browser. Sometimes the cookies that store Session identifiers are set to expire immediately upon closing the browser; these are typically called 'session cookies'. Session identifiers are not only identification tokens, but also authenticators. Once logged

on, users are authenticated based on their credentials and issued session identifiers that will effectively serve as temporary static passwords for accessing their sessions.

LIMITATIONS OF THE EXISTING SYSTEM

1. There were implementation limits on the size and number of cookies that can be stored.
2. Most web sites made use of weak authentication schemes. This was due to lack of central infrastructure and weak uniform security schemes
3. There were many weaknesses in using session ID's
 - Weak algorithm
 - Short length
 - No form of account lock-out
 - Transmitted in the clear
 - Indefinite expiration
 - Insecure retrieval

1.2 PROPOSED SYSTEM AND ITS ADVANTAGES

The proposed system is based on generating session ID's to users only after they are successfully authenticated. To prevent attacks, we propose forceful prevention of logging into chosen session. Some methods to prevent session fixation is

- Binding the session ID to the browser's network address.
- Binding the session ID to the user's SSL client certificate.
- Session destruction by logging out or timeout.
- User logs out thereby destroying current as well as previous sessions.
- Providing Absolute session timeout that prevents attackers from maintaining a trap session.

ADVANTAGES:

The weaknesses with session ID's are solved by sampling session id's, considering the weaknesses while design, transmitting the id over SSL connection. Web applications ignores any session ID provided by the user's browser at login and always generate a new session to which the user will log in if successfully authenticated.

2. SYSTEM REQUIREMENTS ANALYSIS

The purpose of system requirements analysis is to obtain a detailed and thorough understanding of the business need. The requirements are clearly defined and reviewed.

2.1 PROJECT DEFINITION

The project "Prevention of Session Fixation" aims in enhancing secure transactions in web-based applications. The unique feature of this project is providing session ID's to users such that intruders cannot attack it. Our system ensures secure transactions.

2.2 PROJECT PLAN

The requirement phase deals with the sequence of activities in producing the Software Requirement Specification (SRS) document. It must be ensured that all the requirements of the software are elicited and analyzed. In other words the need of the system is identified.

In the problem analysis phase, the current system is analyzed by study of the existing materials, and the changes to be made in the proposed system are decided upon. A clear understanding of the needs of the system must be framed that leads to actual specification.

After the analysis phase, the design phase commences in which the various modules and functionalities are identified. The complete system flow of control and data are identified and depicted in the form of diagrams. Designing aims at how to satisfy the needs of the system. The different modules of the system and the interaction between these modules to produce the desired functionality are identified. During detailed design, the internal logic of each module and their algorithmic design is specified. The major and important decisions are made in this phase.

Coding is the process of translating the design into code. The code developed must be easy to understand. Well-written code can reduce testing and maintenance effort.

In the testing phase, each module is tested thoroughly and finally integrated modules are tested together to ensure the correct working of the entire application. Testing is also done to ensure this application satisfied the specified requirements and set criteria.

WORK	DURATION
1. Feasibility Analysis 2. Abstract Preparation 3. Requirements Gathering	One week
4. Study on existing web applications 5. Study on Tomcat web server	Two weeks
6. Coding Designing a login form and generating a random session ID.	One week
7. Coding- Fixing the user's session with the generated ID	One week
8. Studying the various session fixation possibilities	Two weeks
9. Analyzing the avoidance measures	
10. Implementing the counter measures	One week
11. Generation of Report	One week
12. Implementation of the system	Two weeks
13. Testing of the system	One week

2.3 SOFTWARE REQUIREMENTS SPECIFICATION

2.3.1 PURPOSE

The purpose of this document is to specify the requirements of the project “Prevention of Session Fixation”. It describes the interfaces for the system. The document also bridges the communication gap between the browser and the web server.

2.3.2 SCOPE

SRS forms the basics for agreement between the client and the supplier and what the software product will do. It also provides a reference for the validation of the final project.

Any changes made to the SRS in the future will have to go through formal change approval process. The project gives a way for secured transactions by using enhanced security techniques. This is achieved by binding the IP address to the session ID transmitted so that the data reaches only the authenticated user. Also destroying the sessions would disable the use of those session IDs created during the transaction. The main objective is secure transaction. So the data to be transmitted is hidden in such a way that an intruder cannot obtain the hidden data.

2.3.3 DEFINITIONS

USER

A person who logs into the web to access his requirements.

ATTACKER

A person who is a legitimate user of the system and intrudes into other user's pages.

2.3.4 ABBREVIATIONS

1. SID - Session Identifier
2. SSL - Secure Socket Layer
3. URL - Uniform Resource Locator
4. JSP - Java Server Pages
5. HTTP - Hyper Text Transfer Protocol
6. IP - Internet Protocol
7. JDK - Java Development Kit

2.3.5 GENERAL DESCRIPTION

2.3.5.1 PRODUCT OVERVIEW

This project is to prevent malicious users from gaining illegal access to the various web pages through sessions. The system provides security for transactions between the user and the server. It generates session ID's only if the user is successfully authenticated thereby ignoring any session ID provided by the user's browser at login.

2.3.5.2 USER CHARACTERISTICS

The system will be used mainly in the areas where security is highly needed. It is used to transmit secure information through internet.

2.3.5.3 GENERAL CONSTRAINTS

The application has been programmed to run in the windows platform but can be migrated to other platforms if the need arises. The server however is a multi-platform application.

2.3.6 SPECIFIC REQUIREMENTS

2.3.6.1 INPUTS AND OUTPUTS

The inputs to the system are the login details of any user. In the case of a bank system the account number of the user is also an input. The output of the system is the response to the particular request of the user. It could be the next page of the particular website being accessed by the user.

2.3.6.2 FUNCTIONAL REQUIREMENTS

- The system should be able to authenticate the users by comparing the login details with that in the database.
- The user must be able to proceed without any problem.

2.3.6.3 HARDWARE REQUIREMENTS

Processor	:	PENTIUM IV
RAM	:	128 MB
Hard Disk Capacity	:	20 GB

2.3.6.4 SOFTWARE REQUIREMENTS

Operating System	:	WINDOWS 2000/XP/98
Language	:	Java, JSP
Package	:	JDK 1.5
Server	:	Apache Tomcat

2.3.6.5 PERFORMANCE CONSTRAINTS

The system must execute correctly as long as it gets the correct input from the user and the database formats are intact as defined for the system.

2.3.6.6 SOFTWARE CONSTRAINTS

The application runs on Windows platform. The server requires Java Web services developer pack and jdk1.5 to be installed and running in the server system. The system should not be hosting any other web server application.

3. SYSTEM STUDY

3.1 TOMCAT SERVER:

3.1.1 AN INTRODUCTION TO TOMCAT SERVER

Tomcat server implements the Java Server Pages 1.2 and Servlet 2.3 specifications and includes many features that are useful in the deployment of web modules. Tomcat is written in JAVA thus it needs a java compiler. Therefore, the first step in installation is to download and install a JDK. Tomcat works with JDK1.2. The next steps vary, depending on the operating system on which Tomcat is installed. When Tomcat 5 is started, it creates a set of class loaders that are organized into the following parent-child relationships, where the parent class loader is above the child class loader.

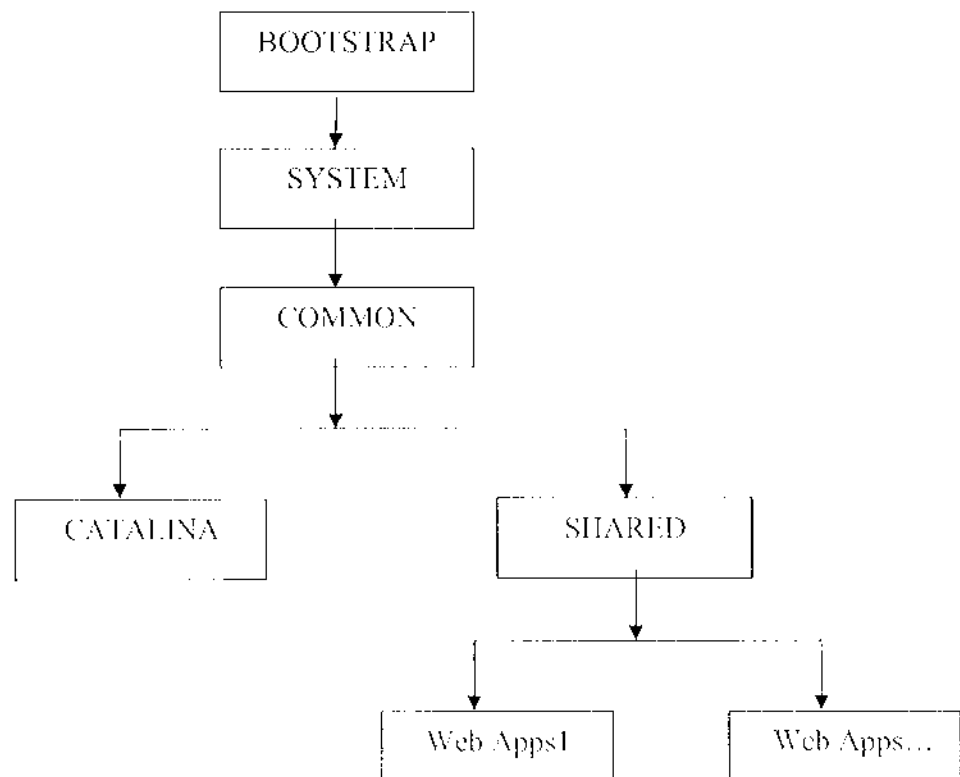


Fig 1:Tomcat Class Loaders

A standard installation of Tomcat makes all of the following APIs available for use by web applications (by placing them in "common/lib" or "shared/lib"):

- * ant.jar (Apache Ant)
- * commons-collections.jar (Commons Collections 2.1)
- * commons-dbcp.jar (Commons DBCP 1.1)
- * commons-el.jar (Commons Expression Language 1.0)
- * commons-logging-api.jar (Commons Logging API 1.0.3)
- * commons-pool.jar (Commons Pool 1.1)
- * jasper-compiler.jar (Jasper 2 Compiler)
- * jasper-runtime.jar (Jasper 2 Runtime)
- * jmx.jar (Sun JMX RI 1.2)
- * jsp-api.jar (JSP 1.2 API)
- * commons-el.jar (JSP 2.0 Expression Language)
- * naming-common.jar (JNDI Context implementation)
- * naming-factory.jar (JNDI object factories for J2EE ENC support)
- * naming-resources.jar (JNDI DirContext implementations)
- * servlet-api.jar (Servlet 2.3 API)

You can make additional APIs available to all of your web applications by putting unpacked classes into a "classes" directory (not created by default), or by placing them in JAR files in the "lib" directory.

3.1.2 INSTALLATION OF TOMCAT SERVER

Installing Tomcat on Windows can be done easily using the Windows installer. Its interface and functionality is similar to other wizard based installers, with only a few items of interest.

- **Installation as a service:** Tomcat will be installed as a Windows NT/2k/XP service no matter what setting is selected. Using the checkbox on the component page sets the service as "auto" startup, so that Tomcat is automatically started when Windows starts. For optimal security, the service should be run as a separate user, with reduced permissions (see the Windows Services administration tool and its documentation).
- **Java location:** The installer will use the registry or the JAVA_HOME environment variable to determine the base path of a J2SE 5 JRE.
- **Tray icon:** When Tomcat is run as a service, there will not be any tray icon present when Tomcat is running. Note that when choosing to run Tomcat at the end of installation, the tray icon will be used even if Tomcat was installed as a service.

The installer will create shortcuts allowing starting and configuring Tomcat. It is important to note that the Tomcat administration web application can only be used when Tomcat is running.

In order to configure JDK logging you should have JDK 1.4+. Tomcat 4.1 can be run on JDK 1.4 using a compatibility package.

The default implementation of java.util.logging provided in the JDK is too limited to be useful. A limitation of JDK Logging appears to be the inability to have per-web application logging, as the configuration is per-VM. As a result, Tomcat will, in the default configuration, replace the default LogManager implementation with a container friendly implementation called JULI, which addresses these

shortcomings. It supports the same configuration mechanisms as the standard JDK `java.util.logging`, using either a programmatic approach, or properties files. The main difference is that per-classloader properties files can be set (which enables easy redeployment friendly webapp configuration), and the properties files support slightly extended constructs which allows more freedom for defining handlers and assigning them to loggers.

Logging can be configured at the following layers:

In the JDK's `logging.properties` file. Check your `JAVA_HOME` environment setting to see which JDK Tomcat is using. The file will be in `$JAVA_HOME/jre/lib`. Alternately, it can also use a global configuration file located elsewhere by using the system property `java.util.logging.config.file`, or programmatic configuration using `java.util.logging.config.class`. In each classloader using a `logging.properties` file. This means that it is possible to have a configuration for the Tomcat core, as well as separate configurations for each webapps which will have the same lifecycle as the webapps.

Tomcat includes a web application (installed by default on context path `/manager`) that supports the following functions:

- Deploy a new web application, on a specified context path, from the uploaded contents of a WAR file.
- Deploy a new web application, on a specified context path, from the server file system.
- List the currently deployed web applications, as well as the sessions that are currently active for those web apps.
- Reload an existing web application, to reflect changes in the contents of `WEB-INF/classes` or `WEB-INF/lib`.
- List the OS and JVM property values.

- List the available global JNDI resources, for use in deployment tools that are preparing <Resource Link> elements nested in a <Context> deployment description.
- List the available security roles defined in the user database.
- Start a stopped application (thus making it available again).
- Stop an existing application (so that it becomes unavailable), but do not undeploy it.
- Undeploy a deployed web application and delete its document base directory (unless it was deployed from file system).

Most commands accept one or more of the following query parameters:

Path - The context path (including the leading slash) of the web application you are dealing with. To select the ROOT web application, specify "/". **NOTE** - It is not possible to perform administrative commands on the Manager application itself.

War - URL of a web application archive (WAR) file, pathname of a directory which contains the web application, or a Context configuration ".xml" file. You can use URLs in any of the following formats:

file:/absolute/path/to/a/directory - The absolute path of a directory that contains the unpacked version of a web application. This directory will be attached to the context path you specify without any changes.

file:/absolute/path/to/a/webapp.war - The absolute path of a web application archive (WAR) file. This is valid **only** for the /deploy command, and is the only acceptable format to that command.

jar:file:/absolute/path/to/a/warfile.war!/ - The URL to a local web application archive (WAR) file. You can use any syntax that is valid for the JarURLConnection class for reference to an entire JAR file.

file:/absolute/path/to/a/context.xml - The absolute path of a web application Context configuration ".xml" file which contains the Context configuration element.

Directory - The directory name for the web application context in the Host's application base directory.

webapp.war - The name of a web application war file located in the Host's application base directory.

3.1.3 DEPLOYMENT

Deployment is the term used for the process of installing a web application (either a 3rd party WAR or your own custom web application) into the Tomcat server. Web application deployment may be accomplished in a number of ways within the Tomcat server.

- Statically; the web application is setup before Tomcat is started
- Dynamically; in conjunction with the Tomcat Manager web application or manipulating already deployed web applications

The Tomcat Manager is a tool that allows URL-based web application deployment features. There is also a tool called the Client Deployer, which is a command shell based script that interacts with the Tomcat Manager but provides additional functionality such as compiling and validating web applications as well as packaging web application into web application resource (WAR) files.

3.1.4 INSTALLATION FOR DEPLOYMENT

There is no installation required for static deployment of web applications as this is provided out of the box by Tomcat. Nor is any installation required for deployment functions with the Tomcat Manager, although some configuration is required as detailed in the Tomcat Manager manual. An installation is however required if you wish to use the Tomcat Client Deployer (TCD).

The TCD is not packaged with the Tomcat core distribution, and must therefore be downloaded separately from the Downloads area. The download is usually labeled *jakarta-tomcat-*x*-deployer*.

TCD has prerequisites of Apache Ant 1.6.2+ and a Java installation. Your environment should define an ANT_HOME environment value pointing to the root of your Ant installation, and a JAVA_HOME value pointing to your Java installation. Additionally, you should ensure Ant's ant command, and the Java javac compiler command run from the command shell that your operating system provides.

- Download the TCD distribution
- The TCD package need not be extracted into any existing Tomcat installation; it can be extracted to any location.

3.2 JAVA SERVER PAGES

Java Server Pages (JSP) technology provides an easy way to create dynamic web pages and simplify the task of building web applications that work with a wide variety of web servers, application servers, browsers and development tools.

Java Server Pages technology allows web developers and designers to easily develop and maintain dynamic web pages that leverage existing business systems [8]. As part of the Java technology family, JSP enables rapid development of web-based applications that are platform-independent. JSP separates user interfaces from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

In its basic form, a JSP page is simply an HTML web page that contains additional bits of code that execute application logic to generate dynamic content. This application logic may involve JavaBeans, JDBC objects, Enterprise Java Beans (EJB), and Remote Method Invocation (RMI) objects, all of which can be easily accessed from a JSP page.

For example, a JSP page may contain HTML code that displays static text and graphics, as well as a method call to a JDBC object that accesses a database; when the page is displayed in a user's browser, it will contain both the static HTML content and dynamic information retrieved from the database.

At first glance, a JSP page looks similar to an HTML (or XML) page--both contain text encapsulated by tags, which are defined between <angle brackets>. While HTML tags are processed by a user's web browser to display the page, JSP tags are used by the web server to generate dynamic content. These JSP tags can define individual operations, such as making a method call to a Java Bean, or can include blocks of standard Java code (known as *script lets*) that are executed when the page is accessed.

Advantages of JSP

Even if you're already content writing servlets for web applications, there are plenty advantages to using JSP:

- JSP pages easily combine static templates, including HTML or XML fragments, with code that generates dynamic content.
- JSP pages are compiled dynamically into servlets when requested, so page authors can easily make updates to presentation code. JSP pages can also be precompiled if desired.
- JSP tags for invoking JavaBeans components manage these components completely, shielding the page author from the complexity of application logic.
- Developers can offer customized JSP tag libraries that page author's access using an XML-like syntax.
- Web authors can change and edit the fixed template portions of pages without affecting the application logic. Similarly, developers can make logic changes at the component level without editing the individual pages that use the logic.

4. DESIGN DOCUMENTS

4.1 INTRODUCTION

A software design is representation of the system of the real world in a format that can be easily understood by both the developers and the users. The diagrams drawn in software design help easy communication between the developers of the various modules of the system and with the users of the system.

In the object oriented paradigm which we have adopted to develop our system, it is easy to make such communication using a standard notation known as UML (Unified Modeling Language). It is an industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems standardized by the Object Management Group. UML simplifies the complex process of software design by using “blueprints” for software construction. Widespread adoption of UML is one of the forces contributing to developer demand for tools that can represent more intentional problem-related information.

4.2 DATABASE DESIGN

The database consists of the user details table.

4.2.1 USER DETAILS

Field Name	Field Type	Field Length	Field Description
Username	Text	50	Indicates the name of the user
Password	Text	10	User assigned keyword

4.2 PROCESS DESIGN

Generally, session fixation attack is a three-step process:

- 1. Session setup:** First, the attacker either sets up a so-called “*trap session*” on the target server and obtains that session’s ID, or selects a – usually arbitrary – Session ID to be used in the attack. In some cases, the established trap session needs to be **maintained** (kept alive) by repeatedly sending requests Referencing it to avoid idle session timeout.
- 2. Session fixation:** Next, the attacker needs to introduce her session ID to the user’s browser, thereby fixing his session.
- 3. Session entrance:** Finally, the attacker has to wait until the user logs in to the target server using the previously fixed session ID and then enter the User’s session.

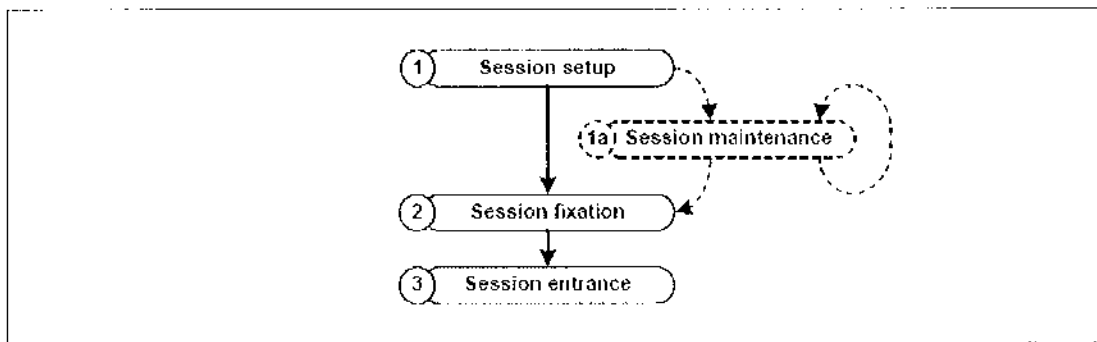


Fig 2. Three steps of session fixation

STEP 1:

SESSION SETUP

The user logs on with a username and password. These details are verified and if valid, a session ID is generated for the particular user. The generated ID is remembered for the remaining sessions. This session will need to be kept alive atleast until the user logs into it. Restarting the web server can destroy all active sessions, requiring the user to login again.

STEP 2:

SESSION FIXATION

SESSION ID IN A HYPERLINK

The attacker needs to trick the user into logging in to the target web server through a look-alike hyperlink that in reality probably comes from another web server. This method is at least as impractical and detection-prone and is included here only for the sake of completeness. In the best case, the attacker could exploit a cross-site scripting vulnerability on the target web server in order to construct a hyperlink (coming from the target server) containing a chosen session ID. However, the attacker managing to trick the user into logging in through a malicious hyperlink could just as well direct the user's credentials to his own web server, which is generally a greater threat than that of fixing his session.

STEP 3:

SESSION ENTRANCE

After the user has logged in to the trap session and before he has logged out, the attacker can enter the trap session and assume the user's identity. In many systems, the attacker will be able to use the session without the user noticing anything suspect. In case the user doesn't log out of the system, the attacker has an opportunity to keep the session alive -- and thereby the access to the user's identity -- for a long time.

STEP 4:

SESSION AVOIDANCE

There is one common denominator to all session fixation attacks and scenarios: the user logs in to a session with an attacker-chosen ID, instead of having been issued a newly generated session ID by the server. Web applications must ignore any session ID provided by the user's browser at login and must always generate a new session to which the user will log in if successfully authenticated. This means that an attacker who isn't a legitimate user of the system will not be able to get a valid session ID and will therefore be unable to perform a session fixation attack.

5.3. Restricting the session ID usage

Most methods for mitigating the threat of stolen session IDs are also applicable to session fixation. Some of them are listed below.

- Binding the session ID to the browser's network address (as seen by the server)
- Binding the session ID to the user's SSL client certificate - very important and often overlooked issue in highly critical applications: *each* server-side script must first check whether the proposed session was actually established using the supplied certificate.
- Session destruction, either due to logging out or timeout, must take place on the server (deleting session), not just on the browser (deleting the session cookie).
- The user must have an option to log out - thereby destroying not just his current session, but also any previous sessions that may still exist (in order to prevent the attacker from using an old session the user forgot to log out from).
- Absolute session timeouts prevent attackers from both maintaining a trap session as well as maintaining an already entered user's session for a long period time

6. PRODUCT TESTING

The system testing deals with the process of testing the system as a whole. This is done after the integration process. The entire system is tested by traversing each module from top to bottom. The verification and validation process are being carried out. The errors that occur at the testing phase are eliminated and a well functioning system is developed.

6.1 UNIT TESTING

It focuses verification effort on the smallest unit of software design, the module. It is also known as module testing. The modules are tested separately. The testing is carried out usually during programming stage itself.

Each and every module is tested separately to check if its intended functionality is met.

6.2 VALIDATION TESTING

Validation is a process of finding out if the product being built is right, i.e. whatever the software product is being developed, and it should do what the user expects it to do. The software product should functionally do what it is supposed to, it should satisfy all the functional requirements set by the user. Validation is done during or at the end of the development process in order to determine whether the product satisfies specified requirements.

After the validation test has been conducted, one of the two possible conditions exists:

- The functions and the performance characteristics confirm to the specification and are accepted.
- Deviation from the specification is uncovered and the deficiency list is created.

6.3 OUTPUT TESTING

After performing the validation testing, the next step is the output testing of the proposed system since no system is useful if it does not produce the required output in the specific format. The outputs generated and displayed by the system under consideration are tested by the users about the formats required by them.

6.4 INTEGRATION TESTING

Here, the tested modules are combined into sub-systems, which are then tested. This is done to test if the modules can be integrated properly, emphasizing on interface between modules. The software was subjected to integration testing and the different modules were linked together and executed.

6.5 SYSTEM TESTING

The system is tested against the software requirements specification to see if all the requirements are met and if the system performs as per the client's expectations. The system is tested as a whole to check for its functionality. Non functional requirements like performance considerations and platform support are checked as a whole.

7. FUTURE ENHANCEMENT

As this project is completely done in JSP, it provides greater flexibility and reusability. Any kind of change can be made to the project without any major change in the underlying functionality. The classes are defined clearly with the necessary access parameters so that they can be modified easily and any additional classes can be added in case of any additional functionality required in the future.

The system can be modified so that it is compatible with the future versions of the server or the programming language .More secure features can be included based on the needs of various web applications.

7. CONCLUSION

To provide designers and implementers with a clear framework, we have given a description of the limitations, requirements, and security models specific to Web client authentication. Web sites have such a large range of requirements that no one authentication scheme can meet them all. Thus the developed system provides the necessary security required for transactions over the web.

APPENDIX – 1

Sample Code

1.Main.jsp

```
<%@page import="java.io.*"%>
<%@page import="java.*"%>
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>SESSION FIXATION</title>
</head>
<body bgcolor="black">
<p align="center"><font size="5"
color="orange"><marquee><H1><b>PREVENTION OF SESSION FIXATION
</marquee></H1></font></p>
<p align="center">&nbsp;</p>
<p align="center">&nbsp;</p>
<p align="center">&nbsp;<b><font size="8" color="blue"><a
href="index.jsp">SESSION FIXATION</a></font></b></p>
</body>
String sslID = (String)request.getAttribute("javax.servlet.request.ssl_session");
System.out.println(sslID);
```

```

%>
<p align="center"><b><font size="8" color="blue"><a href="index1.jsp">SESSION
AVOIDANCE</a></font></b></p>
</body>
</html>

```

2.Index.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<%@page import="javax.crypto.*"%>
<%@page import="java.util.*"%>
<%@page import="java.security.*"%>
<%@page import="java.math.*"%>
<%
    RequestDispatcher dis = request.getRequestDispatcher("intruder.jsp");
    Connection con=null;
    ResultSet rs = null;
    Object obj5=session.getAttribute("username");
    Object obj6 =session.getAttribute("password");
    Object obj7=session.getAttribute("sessionid");
%>
<html>
<title>Prevention of Session Fixation Attack< title>
<body bgcolor="black" >

```

```

<b>
<p align="center"><font size="8" color="yellow">Prevention of Session Fixation
Attack</font></p>
<b>
<form action="index.jsp" name="form" method="post">
<div align="middle">
<fieldset style="width:350px;height:250px">
<legend><font size="5" color="orange">Login</font></legend>
<table align="middle" cellpadding="5" cellspacing="5">
<tr>
<td><font size="5" color="orange"><b>Intruder name</b></font></td>
<td><input type="text" name="uname" size="20"></td>
</tr>
<tr>
<td><font size="5" color="orange"><b>Intruder Password</b></font></td>
<td><input type="password" name="pass" size="20"></td>
</tr>
<tr/>
<tr/>
<tr/> <tr/>
<td align="middle" colspan="1"><input type="submit" value="Enter"></td>
<td align="middle"><input type="button" value="cancel"></td>
</tr>
</table>
</fieldset>
</div>
</form>

```

```

try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    System.out.println("Driver Registered");
    con=DriverManager.getConnection("jdbc:odbc:db1","","");
    Statement stmt = con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE
,ResultSet CONCUR_UPDATABLE);
    rs=stmt.executeQuery("select * from login");
    String uname=request.getParameter("uname");
    String password=request.getParameter("pass");
    while(rs.next())
    {
        if(rs.getString(1).equals(uname)==true)
        {
            if(rs.getString(2).equals(password)==true)
            {
                String sessionid =session.getId();
                Object obj=(Object)sessionid;
                session.setAttribute("username",request.getParameter("uname"));
                session.setAttribute("password",request.getParameter("pass"));
                session.setAttribute("sessionid",obj);
                dis=request.getRequestDispatcher("intruder.jsp");
                dis.forward(request,response);
            }
            else
            {
                if(!(obj instanceof null))

```



```
{
session.setAttribute("username",obj5);
session.setAttribute("password",obj6);
session.setAttribute("sessionid",obj7);
dispatcher=request.getRequestDispatcher("user.jsp");
dispatcher.forward(request,response);
}
}
}
}
}
}
}
}
catch(Exception ex){}
%>
</form>
</html>
```

3.Intruder.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%
RequestDispatcher dispatcher=request.getRequestDispatcher("user.jsp");
%>
<html>
```

```
<head><title>JSP Page</title></head>

<body bgcolor='blue'>

<%

try

{

String hi=request.getParameter("hi");

String uname=null;

String sessionid1;

String pass=null;

Object obj=session.getAttribute("username");

Object obj1=session.getAttribute("password");

Object obj2=session.getAttribute("sessionid");

if(obj!=null)

{

response.sendRedirect("index.jsp");

}

else

{

uname=obj.toString();

pass=obj1.toString();

sessionid1=obj2.toString();

}
```

```

out.println("<html><body><h2><center>intruder Page </center>
</h2><br><br><br>");

out.println("<form method=post action=intruder.jsp><input type=hidden name=hi
value=2><center><input type=submit
value=sendsessionid></center></form></body></html>");

}

if(!(hi==null))

{

session.setAttribute("username",obj);

session.setAttribute("password",obj1);

session.setAttribute("sessionid",obj2);

dis=request.getRequestDispatcher("user.jsp");

dis.forward(request,response);

}

}

catch(Exception exp)

{

System.out.println(exp);

}

%>

< body>

< html>

```

4.Account.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%
RequestDispatcher dis = request.getRequestDispatcher("Account.jsp");
%>
<%
String accountid=request.getParameter("T1");
System.out.println("the value of"+accountid);
if(!(accountid==null))
{
session.setAttribute("account",request.getParameter("T1"));
dis = request.getRequestDispatcher("detail.jsp");
dis.forward(request,response);
}
%>
<html>
<head><title>JSP Page</title></head>
<body bgcolor="black">
```



```
<%@page pageEncoding="UTF-8"%>
<html>
<head><title>JSP Page</title></head>
<body>
<%
try
{
String uname=null;
String sessionid =null;
String pass=null;
Object obj=session.getAttribute("username");
Object obj1=session.getAttribute("password");
Object obj2=session.getAttribute("sessionid");
System.out.println("check 1");
uname=obj.toString();
sessionid=obj2.toString();
pass=obj1.toString();
System.out.println("check");
String sessionid1=session.getId();
System.out.println(sessionid1);
System.out.println(sessionid);
```

```

if(sessionid1==null)
{
response.sendRedirect("index.jsp");
}
else
{
out.println("<a href=Account.jsp>Account.jsp?id=" + sessionid + "</a>");
out.println("password " + pass+"</body></html>");
out.println( sessionid);
}
}
catch(Exception E){}
%>
</body>
</html>

```

6. Detail.jsp

```

<%@ page contentType="text.html"%>
<%@ page pageEncoding="UTF-8"%>
<%@ page session="true" %>

```

```

<%@page import="java.io.*"%>

<%!
Object obj;

%>

<%
String account=null;

RequestDispatcher dispatcher=request.getRequestDispatcher("info.jsp");

%>

<html>

<head>

<meta http-equiv="Content-Language" content="en-us">

<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">

<title>WELCOME TO YOUR ACCOUNT</title>

</head>

<body bgcolor="black">

<%
try
{
obj=session.getAttribute("account");

session.setMaxInactiveInterval(10);

System.out.println("the output is" + obj.toString());

```



```

<p align="center">&nbsp;<b><font size="5" color="white"><a href="B.jsp">
WITHDRAW</a></font>

<p align="center">&nbsp;<b><font size="5" color="pink"><a href="c.jsp">
DEPOSIT</a>

</font></p>

</body>

</html>

```

7.AvoidIndex.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<%@page import="javax.crypto.*"%>
<%@page import="java.util.*"%>
<%@page import="java.security.*"%>
<%@page import="java.math.*"%>
<%
RequestDispatcher dis=request.getRequestDispatcher("Avoidintruder.jsp");
Connection con=null;
ResultSet rs=null;
Object objS=session.getAttribute("username");

```

```

Object obj6=session.getAttribute("password");
Object obj7=session.getAttribute("sessionid");
%>
<html>
<title>Prevention of Session Fixation Attack</title>
<body bgcolor="#ffffcc" >
<b>
<p style="font-size:17" align="center">Prevention of Session Fixation
Attack</p>
<b>
<form action="index1.jsp" name="form" method="post">
<div align="right">
<fieldset style="width:220px;height:175px">
<legend>Login</legend>
<table align="right" cellpadding="2" cellspacing="2">
<tr>
<td>Username</td>
<td><input type="text" name="uname" size="18"></td>
</tr>
<tr>
<td>Password</td>

```



```

String password=request.getParameter("pass");
while(rs.next())
{
if(rs.getString(1).equals(uname)==true)
{
if(rs.getString(2).equals(password)==true)
{
String sessionId=session.getId();
Object obj=(Object)sessionId;
session.setAttribute("username",request.getParameter("uname"));
session.setAttribute("password",request.getParameter("pass"));
session.setAttribute("sessionId",obj);
dis=request.getRequestDispatcher("intruder1.jsp");
dis.forward(request,response);
}
else
{
if(!(obj5==null))
{
session.setAttribute("username",obj5);
session.setAttribute("password",obj6);
}
}
}
}
}

```

```

session.setAttribute("sessionid",obj7);
dis=request.getRequestDispatcher("Avoiduser.jsp");
dis.forward(request,response);
}
}
}
}
}
}
}
}
catch(Exception ex){}
%>
</form>
</html>

```

8.AvoidIntruder.jsp

```

<%@ page contentType="text/html"%>
<%@ page pageEncoding="UTF-8"%>
<%@ page import "java.io.*"%>
<%@ page import "java.util.*" %>
<%

```

```

RequestDispatcher dis=request.getRequestDispatcher("user1.jsp");
%>
<html>
<head><title>JSP Page</title></head>
<body bgcolor='blue'>
<%
try
{
String hi=request.getParameter("hi");
String uname=null;
String sessionid1;
String pass=null;
Object obj=session.getAttribute("username");
Object obj1=session.getAttribute("password");
Object obj2=session.getAttribute("sessionid");
if(obj==null)
{
response.sendRedirect("index1.jsp");
}
else
}

```

```

uname=obj.toString();

pass=obj1.toString();

sessionId=obj2.toString();

out.println("<html><body><h2><center>intruders Page
</center></h2><br><br><br>");

out.println("<form method=post action=intruder1.jsp><input type=hidden name=hi
value=2><center><input type=submit value=sendsessionid>
</center></form></body></html>");

}

if(!(hi==null))

{

session.setAttribute("username",obj);

session.setAttribute("password",obj1);

session.setAttribute("sessionId",obj2);

Random r=new Random(1000);

int a=r.nextInt(1000);

InetAddress adr=InetAddress.getLocalHost();

String str=obj2.toString();

str=str+": "+a+adr;

dis=request.getRequestDispatcher("int.jsp");

dis.forward(request,response);

}

```



```

}
catch(Exception exp)
{
System.out.println(exp);
}
%>
</body>
</html>

```

9.Avoidaccount.jsp

```

<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page import="javax.swing.*" %>
<%@page import="java.util.*" %>
<%
Random r=new Random(1000);
int val=r.nextInt(1000);
int val1=0;
%>
<%_

```

```

RequestDispatcher dis=request.getRequestDispatcher("Account1.jsp");
%>
<%
String accountid=request.getParameter("id");
String accountid1=request.getParameter("T1");
System.out.println("the id1 is "+accountid);
System.out.println("the id2 is "+accountid1);
if(accountid1!=null){
if(!ip.equals("192.168.1.23"))
{
System.out.println("check");
System.out.println("check1");
response.sendRedirect("detail1.jsp?account="+accountid1);
}
}
if(!(accountid == null))
{
System.out.println("-----");
System.out.println("Session Fixation Avoided");
System.out.println("-----");
JOptionPane.showMessageDialog(null, "Session Fixation Avoided", "Session

```



```
<p align="center">&nbsp;</p>
<p align="center"><input type="submit" value="Submit" name="B1"></p>
</form>
<p align="center">&nbsp;</p>
<p align="center">&nbsp;&nbsp;</p>
</body>
</html>
```

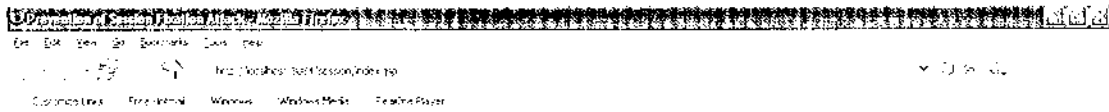
10. Logout.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%
response.setHeader("Cache-Control","no-store");
if (session != null)
{
session.invalidate();
}
response.sendRedirect("index1.jsp");
%>
```

APPENDIX – 2

Screen Shots:

INDEX



Prevention of Session Fixation Attack

Login

Intruder name

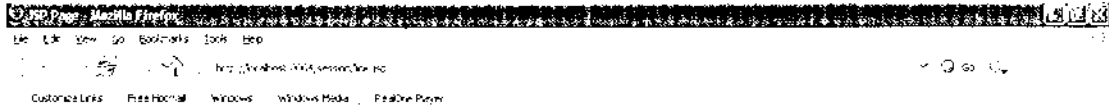
Intruder

Password



2/16/23

LINK



CLICK HERE TO CHECK OUT THE NEW FEATURE OF YOUR BANK
F0C85F704DF4C5F9128EA0D1416D67B4

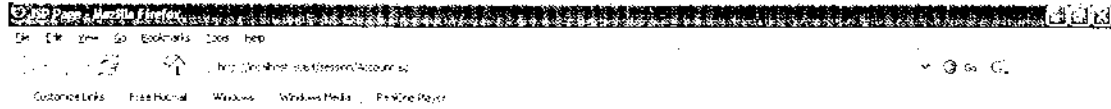
GAMES

ASTROLOGY

E-CARDS

FREE GIFTS

ACCOUNT



WELCOME TO WORLD BANK ACCOUNT

ENTER THE BANK ACCOUNT

1234

5678

Done

LINK BOUND WITH SSL ID AND IP ADDRESS:



CLICK HERE TO LOGIN INTO YOUR BANK ACCOUNT
F1E370EE79ED3B614361CE636444E5F6;487/90.0.0.207

GAMES

ASTROLOGY

E-CARDS

FREE GIFTS

9. REFERENCES:

[1] IETF, »RFC2616: Hypertext Transfer Protocol -- HTTP/1.1«

<http://www.ietf.org/rfc/rfc2616.txt>

[2] IETF, »RFC2109: HTTP State Management Mechanism«

<http://www.ietf.org/rfc/rfc2109.txt>

[3] The Open Web Application Security Project, »Cross Site Scripting«

http://www.owasp.org/asac/input_validation/css.shtml

[4] The Open Web Application Security Project, »Session Hijacking«

<http://www.owasp.org/asac/auth-session/hijack.shtml>

[5] David Endler. »Brute-Force Exploitation of Web Application Session IDs«

<http://online.securityfocus.com/data/library/SessionIDs.pdf>