جامعة فيلادلفيا
نموذج تفويض

أنا ساندريلا إبراهيم كامل محجوب ، أفوض جامعة فيلادلفيا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع :
التاريخ :

Philadelphia University
Authorization Form

I am, Sandrella Ibrahim Kamel Mahjoub, authorize Philadelphia University to supply copies of my thesis to libraries or establishments or individuals upon request.

Signature:
Date:

Using Aspect-Oriented Programming to Secure the Broken Authentication and Session Management on Web Application

By

Sandrella Ibrahim Kamel Mahjoub

Supervisor

Prof. Said Ghoul

This Thesis is Submitted in Partial Fulfillment of the

Requirements for the Master Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia University

May 2009

Successfully defended and approved on _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Examination Committee Signature	**Signature**

Dr,_ _, Chairman. _ _ _ _ _ _ _ _ _ _ _ _ _
Academic Rank: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Dr,_ _, Member. _ _ _ _ _ _ _ _ _ _ _ _ _
Academic Rank: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Dr,_ _, Member. _ _ _ _ _ _ _ _ _ _ _ _ _
Academic Rank: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Dr,_ _, Member. _ _ _ _ _ _ _ _ _ _ _ _ _
Academic Rank: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Dr,_ , External Member. _ _ _ _ _ _ _ _ _ _ _ _ _
Academic Rank: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Dedication

I fully dedicate this thesis to my family; my husband, my sons Bassam and Ibrahim, and my mother in law.

Also I dedicate to my mom and daddy, my brothers and all special friends.

Sandrella Mahjoub

Acknowledgment

I would like to express my regards and appreciation to Prof. Said Ghoul, who has evaluated my work from the beginning and supported me, and to all of those who have helped and encouraged me.

Sandrella Mahjoub

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
AOP	Aspect Oriented Paradigm
CSRF	Cross Site Request Forgery
AOSD	Aspect-oriented software development
JDBC	Java Database Connectivity
J2EE	Java 2 Platform Enterprise Edition
HTML	Hyper Text Markup Language
MITM	Man-in-the-middle
Session ID	Session Identifier
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
XSS	Cross Site Scripting

Abstract

Web applications are becoming more and more popular every day. Many web applications made life easier. We have webmail, online retail sales, online bills payment, flights check-in and status, wikis, multiplayer online role-playing games, and many others.

Due to the increase of web application usage the security become as a most critical aspects which no one can use or trust such application without guarantee security.

Web application concerns contains business concern which takes the principal view, and other concerns as security, logging, tracing, performance ,this all concerns take lower level of importance in the design of a web application. The Aspect-Oriented Programming (AOP) paradigm focuses on the identification, specification and representation of crosscutting concerns and their modularization into separate functional units given more than one concern higher level of importance.

In this thesis, we applied the Aspect-Oriented Programming approach to enhance the security of web applications. That by solving a security problem, the problem is the interception of the client and server connection by a third party to stolen session Id after success login and use the stolen session Id to a legal access to the server response for illegal user.

The Aspect-Oriented Programming approach has already been the subject of several related efforts, which was addressed with the Structured Query Language (SQL) injection and Cross Site Scripting (XSS). These two problems and the session Id stolen problem as a third problem are in the top ten problems for web application security. But the session Id stolen problem was not solved before using the Aspect-Oriented programming approach.

In the thesis we applied the Aspect-Oriented Programming paradigm that introduces such ideas as aspect, joint point models, pointcut and advice to solve a security problem of the web applications. The basic advice applied to solve the problem of the session Id stolen by third part to access the server response as real user, is to make the server to distinguish the origin of request for specific location by using remote address.

As result we have web applications more secure by make a relation between the user request and its remote address as an AOP advice. As AOP concern the security is coded clearly separated from the other concerns making centralized for maintenance. Also as result we have a security API applicable in legacy web application.

Keywords

Application Programming Interface, Aspect-Oriented, AOP Methodology, Broken Authentication and Session Management, Web Application, Web Security.

العنوان

إستخدام البرمجة للجوانب لتعزيز و تحسين أمن إدارة الجلسات في التطبيقات الشبكية

الخلاصة

إزدادت شعبية التطبيقات الشبكية في الآونة الأخيرة وأخذ إنتشارها يزداد يوماً بعد يوم فأضحت أهمية الحماية وضرورة الأمن أكثر إلحاحاً من أي وقت مضى.

تتناول التطبيقات الشبكية عدة شؤون بالإضافة لمتطلبات وحاجات العمل التي تأخذ كل الإهتمام وتطغى على باقي الشؤون؛ مثل الحماية والتسجيل والتتبع والأداء. تركز البرمجة الموجهة للجوانب على فصل الشؤون المتقاطعة بحيث يُمَحوَر كل شأن في وحدة منفصلة مع وصف الأماكن التي يجب على الشأن أن يكون حاضراً فيها، مما يرفع من أهمية باقي الشؤون ويجنب إهمالها.

تتناول هذه الرسالة إستخدام البرمجة الموجهة للجوانب لتعزيز و تحسين الأمن في التطبيقات الشبكة عن طريق تناول المشكلة الأمنية المتمثلة بإعتراض التخاطب بين العميل و الخادم من قبل طرف ثالث لسرقة معرّف الجلسة المولّد عند عملية دخول ناجحة وإستخدام المعرّف للحصول على وصول غير قانوني. تناولت أدابيات سابقة إستخدام البرمجة الموجهة للجوانب لحل المشاكل الأمنية المتمثلة بحقن لغة الاستعلام المهيكلة والحقن عبر الموقع.

ونتيجة لهذا أصبحت التطبيقات الشبكية أكثر أمناً وتولد لدينا واجهة تطبيقات برمجة يمكن إستخدامها في التطبيقات الشبكية الحالية.

Chapter One

Introduction

1.1 Introduction

Internet, web sites and web applications are very popular. Web application (Stuttard, 2008), involves a lot of categories like business contracts, financial notification, medical information, and educational or personal purposes.

Web applications facilitate many businesses. The business process concern for the application always take the high priority in the application over other concerns such as security, logging, tracing which takes lower priority in the web application design. The capability of given higher level of importance to more than one concern at the same time make the Aspect Oriented Programming paradigm applicable in the web application to give other concerns like security a higher importance to make the web application more trustable.

Aspect Oriented Programming paradigm introduces the aspects which consists of pointcut and advice, and introduces join point model which show the execution model for the weaving the application and the aspects.

In the web server there is a session management mechanism which manages the authenticated user accessing to the web application. The session management is responsible to give to the authenticated user his right session access in ever request the user makes to the server in session active time. Sessions gained a unique number or identity by server to each user this identity is called session Id.

In the communication channel between client-server the responses and requests transactions occur. A third parties attacker can intercept the transactions and take the session Id. Once the attacker has a valid session Id he can request from the server as a real user. The interception attack and stolen the session id, and the usage of the session Id by the attacker to access the web server is the problem solved id the thesis. The usage of the session Id which stolen by the attacker to access the server after the user login successfully in the session life time is prevented by applying a session management aspect proposed in the thesis.

The session management aspect in our thesis adds an advice to the web application in certain pointcut points. The advice in ours thesis manipulate an identification process for the logged in user , in the aspect advice the user is related to his location, to give to the web server one more criteria to check if the web request coming is from the real user who made the login operation or coming from elsewhere. The pointcut points for the advice in this thesis are in every request, response process this way the server will not give a response to a user request before check the user request coming location. The session management aspect, uses the AspectJ for weaving, the weaving process make the advice and the web application works as on peace of code, making the output in the user side very comfortable.

1.2 Web Application Common Attacks

As listed by the OWASP (Open Web Application Security Project) (OWASP, 2007) top 10 most critical web application security vulnerabilities in 2007 (Stuttard and Pinto, 2008). The aim of OWASP is to educate developers, designers, architects and organizations about the consequences of the most common web application security vulnerabilities (OWASP, 2007). Cross site Scripting (XSS) (Cannings et.al, 2008), Injection Flaws, Malicious File Execution, Insecure Direct Object Reference, Cross Site Request Forgery (CSRF), Information Leakage and Improper Error Handling, Broken Authentication and Session Management, Insecure Cryptographic Storage, Insecure Communications, Failure to Restrict URL Access.

Two web application security problems was solved with Aspect-Oriented Programming in previous work (Hermosillo et. al, 2007) SQL Injection and Cross Site Scripting, and in these thesis we solved the Broken Authentication and Session Management problem.

1.2.1 SQL Injection

One of the popular ways to web application attack is the SQL Injection attack used by hackers to affect the database of the application via the web interface taking advantage from the web application vulnerability.
As it is defined in (Cannings et. al, 2008), "Attackers use SQL injection to do anything from circumvent authentication to gain complete control of databases on a remote server".
In the SQL Injection attack, the attacker uses the web application to pass SQL commands to the database. The SQL commands can make serious changes in the data on the database or even destroying the data. Here we will show a simple example in a union SQL Injection command, assuming that the command will give a search result, in a website by a given word entered by a user.

Select title from web_pages where keyword ='attack'

If a union command in injected:

Select title from web_pages where keyword ='attack'
Union
Select username, password from application_users

Here the search result will appear, and also the usernames and passwords will appear to the attacker.

1.2.2 Cross Site Scripting

Another popular type of web application attack is the Cross site Scripting. The main idea of the cross site scripting is to place a script like JavaScript, VBScript, or other browser-accepted scripting languages into vulnerable web application; simply the browser believes that the script came from the web application. This type is used to keep cookies containing information identifying users, or to provide some data to the attacker.

The classic Cross Site Scripting (XSS) attack is a reflected HTML injection attack whereby a web application accepts user input in an HTTP request (Cannings et. al, 2008).

As an example for the XSS, assuming that there is an “insert your email “field for a web application. If the attacker enters a script instead of the email address as:

```
<script>
        Alert(“ATTAAAAAAK !!”);
</script>
```

So every time there is a call for the email addresses as result the script will be called .If there is a loop of thousand in the script it will run thousand times.

1.2.3 Broken Authentication and Session Management

Authentication and session management includes all features of dealing with user authentication and managing active sessions. Authentication can be considered as a process of confirming the correctness of someone’s identity. A session management mechanism will depends on the authentication of the user to give this user his right session access in ever session is active.

A session is marked by a unique number or identity given by the web server to the user. This identity can be called as session Id. With the session Id the web server can recognize the user who is making the request so the server can give the correct response to correct user. The session Id will stay with the user as marker for the time in which the session is active. Like that the users do not need to authenticate every time their request from the server.

The requests and responses transactions between client-server communications can be attacked by a third parties attacker, who intercept the transactions and take the session Id. With this session Id on hands the attacker can request from the server as a real user, and the attacker will get the responses from the server as the real user.

As an example we have a user ‘A’ who authenticates successfully on a web application and the web server gives to user ‘A’ a session Id equal “H976NL09824H860004GXW”. Every time a request comes with the session Id ‘H976NL09824H860004GXW’ a ‘Hello A!’ will appears to the user.

In the same example an attacker ‘Z’ intercept the client-server transaction and catch the session Id “H976NL09824H860004GXW”, than the attacker “Z” requests from the server using the stolen session id, the server will respond to “Z” with “Hello A!” as response be to user “A” who makes the request.

1.3 Aspect-Oriented Paradigm Solutions to Common Attacks

Aspect Oriented Programming paradigm was used in several works to solve problems like logging, tracing and security. In java there is the AspectJ for the practical aspect-oriented supporting the modular implementation of a range of crosscutting concerns (Kiczales et. al, 2001).

AspectJ language makes the design of crosscutting concerns possible in a modular way. Because of the AspectJ modularity it is used in the implementation of the aspects for the web application security and others everyday situations such as logging, policy enforcement, resource pooling, business logic and thread-safety (Laddad, 2003). Modular programming (Boudreau et. al, 2007) and aspect-oriented programming are two approaches software engineering used to help in application design (Lieberherr et. al, 2003).

One of the solutions is by using AOP crosscutting function in complex software taking advantage of the new software development approaches (Hermosillo et. al, 2007). The work done in (Hermosillo et. al, 2007) is an experimentation of the advantages for chaining security policies at run time , the experimentations was done in the first two insecurities in the design and implementation of web application ,SQL Injection and XSS .

The solution that is presented in (Hermosillo et. al, 2007) which give a security aspect in a web application server. Their work uses the aspect to detect SQL injection and XSS attacks in users' requests to web application, and from the web server to a database server. It allows the interception of all database accesses and the validation of them before dangerous information is stored.

In the work of (Hermosillo et. al, 2007) an Aspect-Oriented Programming solution was presented in on web security for SOL Injection and XSS attach. In these work the authors used an aspect for validate the SQL Injection by setting these aspect in the web application server, in these case the aspect checks the SQL validation in the request from the database. Also the authors had done an aspect to validate the XSS attacks, that aspect checks for the scripts that may come with the request from the user to the web application server. The authors used web servers as Tomcat and JBoss in their work.

In the work of Kawachi and Masuhara (Kawauchi and Masuhara, 2004), the authors propose an aspect to detect crosssite scripting. Their approach is based on validate the parameters by replacing special characters by quoted ones, in the input data submitted by users to web applications.

1.4 Problem Definition

Vulnerability in web applications exposes the data, user's transactions and the system to hackers who easily can have an illegal access to data, application, or server machine making damage on the system, or take advantage from the acquired information.

The importance of security is not less than the business or performance for the web application, so here in this thesis we will give a higher level of importance for the security in the web application, without change the business or performance priority level, by using the Aspect-Oriented Programming paradigm to solve a problem in the web security making the web Application more secure preserving privacies to the application users.

The security problem scope solved in the thesis is the session life time, after the users make a successful login in the application until end of session. Within this time is attackers intercept the transaction between the client and the server to stolen session Id. The attacker can stole the session Id by sniffing in the network and take information from the client , server transactions .Once the legal session Id is with the attacker than the session Id can be used to a legal access to the server response for illegal user, this problem is called broken authentication and session management.

Aspect Oriented Programming Paradigm was used to apply the remote address, from the request header to differentiate the real user accessing the server, as an aspect advice in a session management aspect. Preventing illegal user's to requests from the server.

1.5 Thesis Outline

In our thesis we have:

- Completed the work presented on (Hermosillo et. al, 2007) which worked in AOP and web application security solved the SQL Injection and XSS attacks , our complement to their work come by solving the broken authentication and session management security problem, which was not yet solved. Our solution gives us the advantage to not change in the current code of legacy web applications, when add a new security concern as an aspect. Our solution proposed is a session management aspect which adds a check for the request user location.

- Proposed an AOP methodology, guiding web applications developers in securing their new or legacy application, against broken authentication and session management attacks. The methodology consists of some steps that guide the usage of AOP paradigm in a new or legacy application to apply a security concern. In the methodology a new security advice is used to solve the broken authentication and session management problem. This makes an enhancement in the web application security problems.

- Designed an API to support the proposed methodology. In this thesis we provide a programming interface which shows the applicability of the proposed methodology. The programming interface was used to help in the programming interpretation for the methodology, which was applied in a study case in this thesis to get results and better understand the thesis contribution.

Chapter Two

Literature Review

2.1 Web Application

Web applications (Jendrok et.al, 2006) are very popular and their popularity increases day by day. Many web applications functionalities really facilitate life. We have webmail, online retail sales, online bills payment, check in flights and status, wikis, multiplayer online role-playing games and many others.

In web application architecture (Figure 2.1), users obtain information in the browsers using the application server. The application server interacts with clients and database servers. Browsers used by the users send request to the web server directly, if there are no validations or poor ones, some unexpected behavior may occur. Web server forwards the non valid parameters to the database server in the database server.

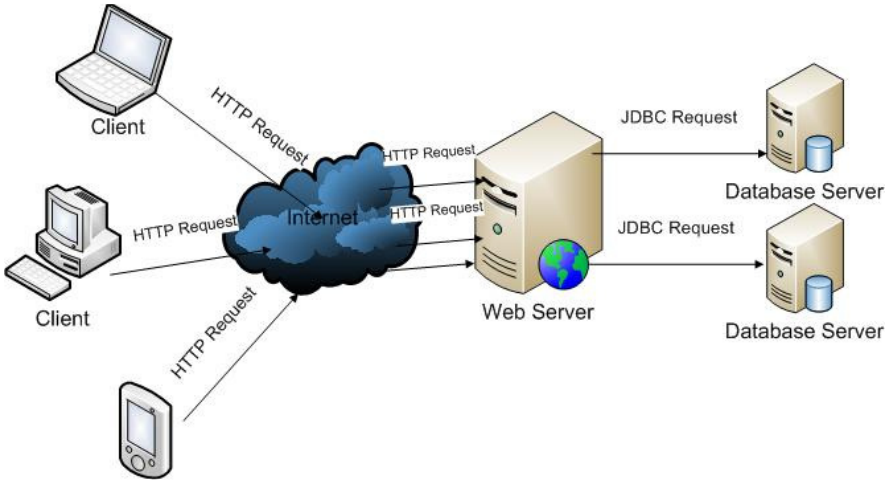

Figure 2.1 Web applications Architecture

Web applications mainly contain three parts:

- Part One: A bunch of clients using browser to access the web application via internet.
- Part Two: The data offered by the web application. This data is stored in a server named data server or database server. The data is showed to the client on the browser via the web application in the web server.
- Part Three: The web server. In the web server machine there is a software responsible for running the application code. There are many kinds of application server software some from vendors and other open source like Oracle Application Server (Stackowiark, 2004), IBM Web Server (Sadtler et.al, 2005), Tomcat (Chopra et.al, 2007) (Chopra et.al, 2004), JBoss (Davis et.al, 2005), etc.

2.1.1 Web server session management

In human-computer interaction, session management is the process of keeping track of a user's activity across sessions of interaction with the computer system.

Hypertext Transfer Protocol (HTTP) is stateless (Wong, 2000): a client's computer running a web browser must establish a new Transmission Control Protocol (TCP) (ISO/IEC, 1997) network connection to the web server with each new HTTP GET or POST request. The web server, therefore, cannot rely on an established TCP network connection for longer than a single HTTP GET or POST operation. Session management is the technique used by the web developer to make the stateless HTTP protocol support session state. For example, once a user has authenticated himself to the web server, his next HTTP request (GET or POST) should not cause the web server to ask him for his account and password again. One of the methods used to accomplish this is the HTTP cookie , (Wong, 2000).

The session information is stored on the web server using the session identifier (session ID) generated as a result of the first (sometimes the first authenticated) request from the end user running a web browser. The "storage" of session IDs and the associated session data (user name, account number, etc.) on the web server is accomplished using a variety of techniques including, but not limited to: local memory, flat files, and databases (Dave et.al, 2006).

2.2 Web Security

2.2.1 Web Security Problems

Web security, is a set of procedures, practices, and technologies for assuring the reliable, expected operation of web servers, web browsers, other programs that communicate with web servers, and the surrounding Internet infrastructure. Unfortunately, the complete scale and complexity of the Web makes the problem of web security dramatically more complex than the problem of Internet security in general (Garfinkel, 2001).

In following we can see three parts where a web security problem can occurs:

- *In the web server and the data*
  Security in the web server is related to the functionality of the operations supposed to be done by this server and the information in the server. The access to the web server should be to authorized persons.

- *With the information that travels between the web server and the user*
  It is important to assure that the link between the user and the web server cannot be easily disrupted. Check that the information that the user supplies to the web server as (usernames, passwords, financial information, the names of web pages visited, etc.) cannot be read, modified, or destroyed by any third parties. Protection for the information that flows back from the web servers to the users is also needed.

- *The end user's computer and other devices that people use to access the Internet*

The end user's computer needs to be secure as a part of the chain. Users need to run their web browsers and other software on a secure computing platform that is free of viruses and other hostile software. Also users need to protect their privacy and personal information, on their own computers or in their online services.

2.2.2 Session hijacking

The term session hijacking (Stuttard and Pinto, 2008) refers to the exploitation of a valid computer session - sometimes also called a session key - to gain unauthorized access to information or services in a computer system. In particular, it is used to refer to the theft of a magic cookie used to authenticate a user to a remote server. It has particular relevance to web developers, as the HTTP cookies used to maintain a session on many web sites can be easily stolen by an attacker using an intermediary computer or with access to the saved cookies on the victim's computer.

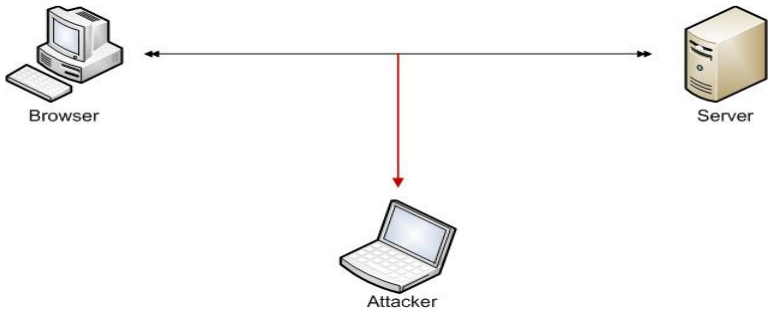

Figure 2.2 Man-In-The-Middle is attack

HTTP cookies, more commonly referred to as Web cookies, tracking cookies or just cookies, are parcels of text sent by a server to a Web client (usually a browser) and then sent back unchanged by the client each time it accesses that server. HTTP cookies are used for authenticating, session tracking (state maintenance), and maintaining specific information about users, such as site preferences or the contents of their electronic shopping carts. The term "cookie" is derived from "magic cookie," a well-known concept in UNIX computing which inspired both the idea and the name of HTTP cookies.

In the Man-In-The-Middle (MITM) attack (Figure 2.2) we can say that type can only be successful when the attacker can impersonate each endpoint to the satisfaction of the other. Most cryptographic protocols including some form of endpoint authentication specifically try to prevent MITM attacks. In this kind of attack the attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances (for example, the owner of a public wireless access point can in principle conduct MITM attacks on the users).

Many web sites allow users to create and manage their own accounts, logging in using a username and password (which may or may not be encrypted during transit) or other authentication method. In order that the user does not have to re-enter their username and password on every page to maintain their session, many web sites use session cookies: a token of

information issued by the server and returned by the user's web browser to confirm its identity, This kind of web application is vulnerable for the MITM attack.

If an attacker is able to steal this cookie, they can make requests themselves as if they were the real user, gaining access to privileged information or changing data. If this cookie is a persistent cookie, then the impersonation can continue for a considerable period of time. Of course, session hijacking is not limited to the web; any protocol in which state is maintained using a key passed between two parties is vulnerable, especially if it's not encrypted.

Use of a Secured Identification card, or other token based secondary authentication is useless as protection against hijacking, as the attacker can simply wait until after the user authenticates, then hijack the session (Dittrich, 1999).

There are four main methods used in session hijack (Stuttard and Pinto, 2008). These are:

- Session fixation, where the attacker sets a user's session id to one known to him, for example by sending the user an email with a link that contains a particular session id. The attacker now only has to wait until the user logs in.
- Session side-jacking, where the attacker uses packet sniffing to read network traffic between two parties to steal the session cookie. Many web sites use SSL encryption for login pages to prevent attackers from seeing the password, but do not use encryption for the rest of the site once authenticated. This allows attackers that can read the network traffic to intercept all the data that is submitted to the server or web pages viewed by the client. Since this data includes the session cookie, it allows him to impersonate the victim, even if the password itself is not compromised (Dittrich, 1999), (Stuttard and Pinto, 2008).Unsecured Wi-Fi hotspots are particularly vulnerable, as anyone sharing the network will generally be able to read most of the web traffic between other nodes and the access point.
- Alternatively, an attacker with physical access can simply attempt to steal the session key by, for example, obtaining the file or memory contents of the appropriate part of either the user's computer or the server.
- Cross-site scripting, where the attacker tricks the user's computer into running code which is treated as trustworthy because it appears to belong to the server, allowing the attacker to obtain a copy of the cookie or perform other operations.

2.3 Aspect-Oriented Programming Paradigm

Aspect Oriented Programming (AOP) is a programming paradigm that increases modularity of applications, by allowing the separation of cross-cutting concerns (Kiczales et.al, 1997).

When software is designed the architects first concentrate on the primary core functionality, which in a business application is basic business logic concern. The software applications also involve other features as logging, authorization, persistence, and other elements. All concerns are with importance for applications. A system with a good number of concerns that extent multiple

modules are called crosscutting concerns. Aspect-oriented programming manages these crosscutting concerns.

In the beginning of the AOP in the Xerox Palo Alto Research Center in 1996 (OWASP, 2007), a researcher Gregor Kiczales and his team choose the Tomcat servlet engine in one of the experiences that motivate the classification of AOP. One of the first practical implementations of AOP was done in the late 1990s. AspectJ is an implementation of AOP based on Java, but there are implementations of AOP for other languages, ranging from AspectC for C to Pythius for Python, that apply the same concepts that are in AspectJ to other languages.

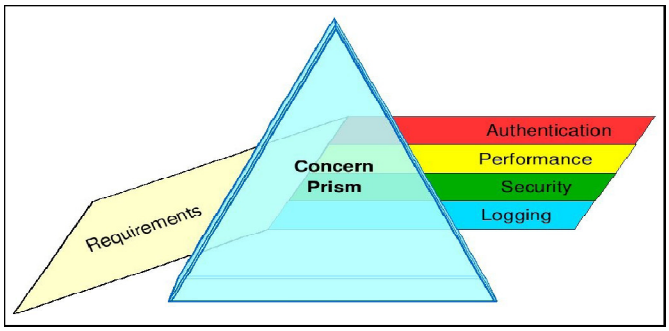

Figure 2.3 Prism comparisons for concern separation (Hemosillo et.al,, 2007)

A comparison of how the different concerns can be separated from the requirements using AOP is how a prism separates a light beam into a spectrum of colors (Figure 2.3).

Traditional software development has focused on decomposing systems into units of primary functionality, while recognizing that there are other issues of concern that do not fit well into the primary decomposition. The traditional development process leaves it to the programmers to code modules corresponding to the primary functionality and to make sure that all other issues of concern are addressed in the code wherever appropriate. Programmers need to keep in mind all the things that need to be done, how to deal with each issue, the problems associated with the possible interactions, and the execution of the right behavior at the right time. These concerns span across the primary functional units within the application, and often results in serious problems faced during the application development and maintenance. The distribution of the code for realizing a concern becomes especially critical as the requirements for that concern evolve, a system maintainer must find and correctly update a variety of situations.

Looking at the Object-Oriented programming (Kiczales et.al, 1997) entered the majority of software development, visualizing systems as groups of entities and the interaction between those entities, which allowed them to deal with larger, complicated systems and develop them in less time than ever before. There still a problem with the OO programming which is basically static, meaning that in the requirement changes it will cause a big delay on development timelines.

Aspect-Oriented Programming solves the problem of the OO programming with the characteristic to dynamically change the static OO model to generate systems that accept smoothly new requirements, as the object in the real world can change their states during their lifecycles, and the application also can accept new requirements (O'Regan, 2004).

Aspects can be applied (the term used by the AOP community is woven) at compile time or at runtime. Experience has shown the difficulty of writing crosscutting functions such as security (Viega et.al, 2001).

A motivation for aspect-oriented programming languages stem from the problems caused by code scattering and tangling. The purpose of Aspect-Oriented Software Development is to provide systematic means to modularize crosscutting concerns.

The implementation of a concern is scattered if its code is spread out over multiple modules. The concern affects the implementation of multiple modules. The implementation of a concern is tangled if its code is intermixed with code that implements other concerns. The module in which tangling occurs is not cohesive.

Aspect-oriented software development considers that code scattering and tangling are the symptoms of crosscutting concerns. Crosscutting concerns cannot be modularized using the decomposition mechanisms of the language (object or procedures) because they inherently follow different decomposition rules. The implementation and integration of these concerns with the primary functional decomposition of the system causes code tangling and scattering.

Aspect-oriented programming provides a mechanism to address each concern separately with minimal combination. This results in modularized implementation even in the presence of crosscutting concerns. Such implementation results in a system with much less duplicated code. Because the implementation of each concern is separate, it also helps avoid code clutter. Modularized implementation results in an easier-to-understand and easier-to-maintain system.

2.3.1 Concepts Terminology

Figure 2.4 Aspect-oriented programming terminologies

Aspect-oriented programming introduces some concepts (Figure 2.4), these concepts are the crosscutting concerns, aspects, pointcut, advice, join point models and weaving:

- **Crosscutting concerns:** A concern in software application is a functionality module in the software, for example software example if we have a banking application, the core concerns are the bank transaction make by each customer. The concerns that are common to many of the core modules have features such as logging, authorization, and persistence. These system-wide concerns that span multiple modules are called crosscutting concerns. Aspect-oriented programming (AOP) manages these crosscutting concerns.

- **Advice**: This is the additional code that you want to apply to your existing model. For example in a database logging on each update statement, here every time an update statement is done in the database, a code is executed making a statement to write the update statement executed in the database, the date of the update statement execution and the user who executed the update statement in the database .This log code process is wrote in the advice.

- **Pointcut:** This is an AOP term given to the point of execution in the application at which crosscutting concern needs to be applied. Pointcuts are the points in the program where the AOP advice should be introduced. In the example of the database logging on each update statement the pointcut is the update statement, so the point to apply the advice code is the update statement.

- **Aspect**: Aspect-oriented languages provide explicit support for localizing crosscutting concerns into separated modules, called aspects. An aspect is a module that encapsulates a crosscutting concern. We can say that it is the combination of the pointcut and the advice. For example the advice for saving information about the database update statement and the pointcut to save the information which is the update statement, these two together are the aspect for a database logging in the update statement example.

- **Join point models**: Join points are points in the runtime execution of the system, such as method calls, where aspects inject behavior through advice bodies. A join point is a point in the execution of the program, which is used to define the dynamic structure of a

crosscutting concern. It is a particular point in the region defined by a pointcut. Some of the join point models can be before or after execution, can be before or after call or around it. So the joint point can be for the logging example after each execution of the update statement, but for a validation example the join point can be before each calling for the update statement.

- **Weaving**: The injecting the advice presented in aspects into the specified join-points associated with each advice - provides the final challenge of any AOP solution. The weaving is done for example by the AspectJ compiler and the result is as java classes. Here the advice is placed in the places pointed by the pointcut in the application to give the output to end user as it is expected.

A pointcut is a program element that picks out join points and exposes data from the execution context of those join points. Pointcuts are used primarily by advice.

The join point model of an aspect-oriented language defines the types of join points that are supported by the aspect-oriented language and the possible interaction points between aspects and base modules.

Since join points are dynamic, it may be possible to expose runtime information such as the caller or callee of a method from a join point to a matching pointcut. Nowadays, there are various join point models around and still new under development. They heavily depend on the underlying programming language and AO language.

A method call join point covers the actions of an object receiving a method call. It includes all the actions that compose a method call, starting after all arguments are evaluated up to return.

Many AOP languages implement aspect behavior by weaving hooks into join point shadows, which is the static projection of a join point onto the program code.

Figure 2.5 Examples of join points

The advice code runs in a join point that is selected by a pointcut. This advice can execute before, after, or around the join point. Around advice can modify the execution of the code that is at the join point, it can replace it, or it can even bypass it (Figured 2.5) depend on the join point model selected for the suitable situation. Using an advice, we can log a message before executing the code at certain join points that are spread across several modules.

2.4 Web Application Security with AOP Approach

Aspect-Oriented Programming is a good proposed technique that uses concerns (Kiczales et.al, 1997). AOP has been proposed as a technique for improving concerns separation in software systems and for adding crosscutting functionalities without changing the business logic of the software. AOP provides specific language mechanisms that make it possible to address concerns, such as security, in a modular way. AOP languages and tools can be applied at compile time or at runtime. This way, the security issue in a software system can be addressed.

A guide to aspect-oriented programming and the AspectJ language (Laddad, 2003) this work provides examples introductions to AOP and AspectJ will help learning or advance knowledge in AspectJ. Examples of everyday situations in which AspectJ solutions can be applied, such as logging, policy enforcement, resource pooling, business logic, thread-safety, authentication and authorization, and transaction management are provided.

The capability of the Aspect-Oriented programming languages to decomposing systems into modules and composing modules into systems made the AspectJ selected by (Griswold et.al, 2006) in their work. In the (Griswold et.al, 2006) work they used the aspect-oriented programming for better modularization of the aspects and advise code without limiting the number of advises. The authors used a pointcut descriptor (PCD) that declares specific set of points. Their approach uses XPIs and it allows for their separate and parallel evolution and produces a better correspondence between programs and designs.

Applying security with aspects, you can modularize your security code in one place, apply the security policies transparently to a large degree, and apply security to an application where the concern was not originally part of the design.

Security can be considered as an important cross-cutting concern to web application. Security characteristics rarely have anything to do with the simple business logic of an application. They are often disturbing. Unfortunately security is the last thing to be applied to a piece of software.

Security with aspect has been the subject of several works (Bostrom, 2004) (Laney et.al, 2004) (Huang et.al, 2004) (Verhanneman et.al, 2004) (De Win et.al, 2005) (De Win et.al, 2003). Most implementations of these studies apply the AspectJ (Laney et.al, 2004) (Huang et.al, 2004).

In the work of Gabriel Hermosillo and Roberto Gomez (Hemosillo et.al, 2007) they demonstrate their approach for writing a security aspect in a web application server. These web security aspects worked on the SQL injection attack and XSS attacks. Their work detects SQL injection

and XSS attacks in requests to a web application and from this web server to a database. They apply their advice with the database validation before the data request get in the database server. As an AOP approach the aspect code which contains the security is completely separated from the application code. By that the original code for the application was not changed.

Working with a database encryption to make information more confidential was in the paper work of (Bostrom, 2004) .They used the AOP approach and AspectJ to implement the crosscutting for database encryption. They provide the encryption for the database as a separated aspect. That makes developers to focus their efforts on business logic for the application. The encryption is to be dependent on which column is written or read.

In the work of (Laney et.al, 2004), the contribution was in showing how aspects could be used to evolve legacy code. They have shown a solution for security problems related to "message tampering" attacks. Their goal was to add digital signatures to the system to enhance the authorization mechanisms. The work was based on the use of a study case encompassing multiple servers in a prototype home banking application. They focus on the aspect reuse one of the AOP approach advantages.

In the work of (Huang et.al, 2004), they focus on providing practical reusable components at the level of aspects in AspectJ. They implemented a security aspect library, and they give an example of application in which the security aspect library was used. The main advantage in the security aspect library proposed was the reusability and generality. The distinct differences between security-related contexts in some applications make the generality of security aspects not easy. Other advantage is that the security aspects are easy to modify and maintain.

In the (Verhanneman et.al, 2004) paper is proposed a new approach for engineering access control into application. The approach is based on the concepts of access interface and view connector. In the access interface they show the abstract view for an application. The application view connectors maps the application concepts to the domain concept represented in the access interface and assign object instance to domain. A clear separation between the different roles, the security officer, deplorer and security module provider, is supplied by the terms of the access control aspect development and lifecycle, thus step offers the fulfillment of the access control requirements with the integration of access control logic in the application.

In the work of (De Win et.al, 2003), the authors made experiences with a study case that support AOSD techniques. Their goal is to improve modularity and understandability. The modularity was showed by the clear separation of the security using the capability of the AspectJ tool. The modularization of crosscutting concerns improves the understandability and analyzability of security.

2.5 Conclusion

The following stated security problem was not yet solved by AOP approaches: broken authentication and session management, malicious file execution, cross site request forgery (CSRF), insecure direct object reference, information leakage and improper error handling, insecure cryptographic storage, insecure communications, failure to restrict URL access.
In this thesis we proposed a solution to broken authentication and session management problem.

Chapter Three

An Aspect-Oriented Methodology

3.1 Methodology Philosophy
3.2 Methodology Model
3.3 A Web application Model
3.4 Case Study
3.5 Obtained Results

3.1 Methodology Philosophy

In the methodology philosophy we apply an AOP methodology to a legacy application which is not secure against broken authentication and session management or to a new application to get as result a secure web application against broken authentication and session management

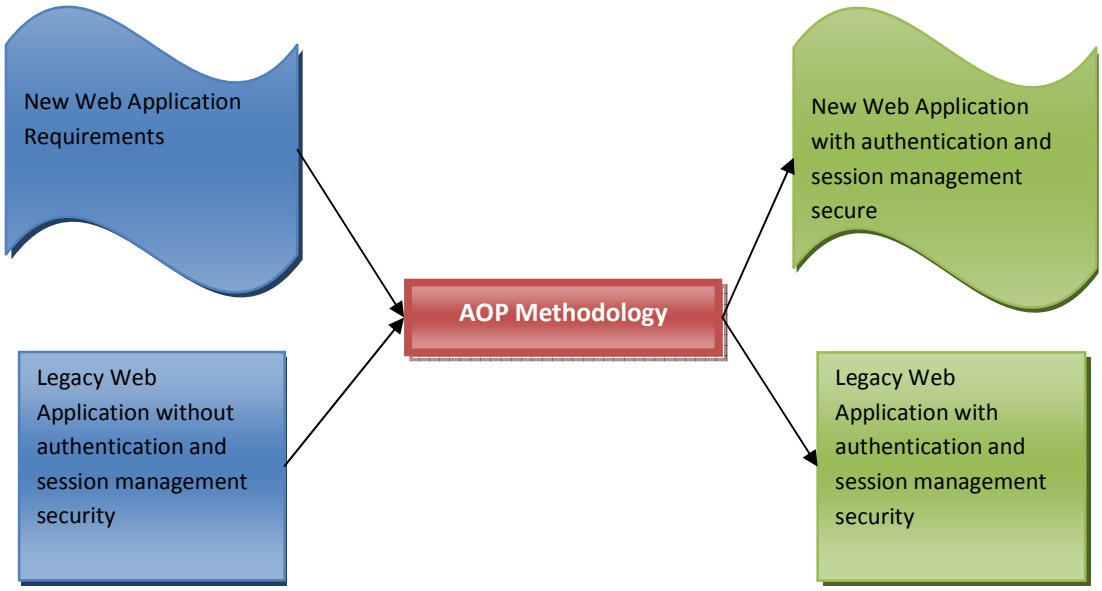

Figure 3.1 Methodology Philosophy

The methodology philosophy proposed in this thesis (Figure 3.1) has the following:

- New Web Application Requirements: is a call for a new application needed to be developed given some requirements to be in the application.
- Legacy Web Application without authentication and session management security: An existing and working application that is defenseless in front of broken authentication and session management attack.
- AOP Methodology: Is our AOP methodology used for web security, to deny broken authentication and session management attack.
- New Web Application with authentication and session management secure: is a new application that used the AOP Methodology for security.
- Legacy Web Application with authentication and session management security: An existing working application that applied the AOP Methodology for security to be with secure authentication and session management, without change in the exit code.

3.2 Methodology Model

In the following we summarize the methodology model using pseudo code formalism (Figure 3 .2).

```
Begin

    Create a map to store the session Id remote Address; [1]
    Select the appropriate pointcuts in the web application for the session management; [2]
    Build the Advise body for Web Application; [3]
    Set the proper Advice in for the proper join point; [4]

End
```

Figure 3.2 General Methodology Model

This methodology is supported by an Application Programming Interface (API) including **SessionMap** (supporting the step 1), and **Pointcut_and_advice** (supporting the steps 2,3 and 4).

The **SessionMap** (Figure 3.3) creates the Map and stores the session Id (key) and the remote address (Value). The Map used on the implementation of the sessionMap in this thesis is the hash map in the java API. This class has three functions , which deals with the values of the Map.

There are three functions (put, remove, get) to manipute the operations on the SessionMap like delete , insert , update and retrive the data.

- The function "*put*." That inserts the key and value. This function does not return anything (void), and take two parameters from type String the (key) and the (Value).
- The function "*remove"* .That removes the value and key. This function does not return anything (void), and take one parameter from type String the (key).
- The function "*get"*. That returns the value of a given key. This function returns a String value (remote address or empty), and take one parameter from type String the (key).
- The "key" value in the map is the session Id given to the user after the success login.
- The "Value" in the map is the remote address for the user who makes the request.

```
public class SessionMap {
        private static Map map = Collections.synchronizedMap(new HashMap());
        public static void put(String key, String value)
        public static void remove(String key)
        public static String get(String key)
}
```

Figure 3.3 SessionMap Class in Java.

The **Pointcut_and_advice,** (Figure 3.4) supports the pointcut which includes a predefined Advice body.

```
public aspect ServletSecurityAspect {

        public pointcut getHTTPRequest(HttpServletRequest request,HttpServletResponse response) :

        before(HttpServletRequest request, HttpServletResponse response)

        public pointcut postHTTPRequest(HttpServletRequest request,HttpServletResponse response) :

        after(HttpServletRequest request, HttpServletResponse response)

}
```

Figure 3.4 Pointcut_and_advice aspect in AspectJ

The pointcut (Figure 3.4), fix where to apply advice.

We have two kind of pointcuts:

- *Pointcut.Post (Figure 3.5)*: For a join point after the login, because in this method the authentication for the user is done. So after the authentication the advice in the aspect is joined to the application method code.
- *Pointcut.Get (Figure 3.6)* : For a joint point before any result showing to the user. This join point will process the advice.

```
public pointcut postHTTPRequest(HttpServletRequest request,HttpServletResponse
response) :
                        execution(* doPost
                                    (HttpServletRequest,
                        HttpServletResponse))
                        &&
                        args(request, response);
```

Figure 3.5 postHTTPRequest pointcut

```
public pointcut getHTTPRequest(HttpServletRequest request,HttpServletResponse response) :
                    execution(* doGet
                                (HttpServletRequest,
                    HttpServletResponse))
                    &&
                    args(request, response);
```

Figure 3.6 getHTTPRequest pointcut

The predifined **Advise body** (Figure 3.7), processes the advice security code and manages the session using the Session map. It is the check criteria for the aspect. This Advice body can be used in any web application using java . In the advice body the security attack problem is treated by enhancing the session management (associating the Remote Address with the session id).

- The advice makes the application continue the normal behavior if the request in coming from real client.
- The advice changes the application response in the case if requests are coming from wrong place.

Its process is modeled in (Figure 3.7) .The prosesed model was done here as a psoudo code model. The model give steps on how they the security Aspect-Oriented model wroks with web applications.

```
Begin
    Set values in the variables
        Session Id
        Current Remote Address
        Stored Remote Address
    Compare Stored Address
            If Stored Remote Address is null then
            //This is the first time request of the given session; Associate the Session //Id with the remote Address.
            Map.put (Session Id , Current Remote Address )
            Else
              If Stored Remote Address is not equal the Current Remote Address
            // There is a security attack here!
            Set the response HTTP 403
End
```

Figure 3.7 Advice body psoudo code model

3.3 A Web application Model

Applying the AOP methodology to a new or legacy web application; leads to the new or legacy authenticate secure application modeled as it follows (Figure 3.8)

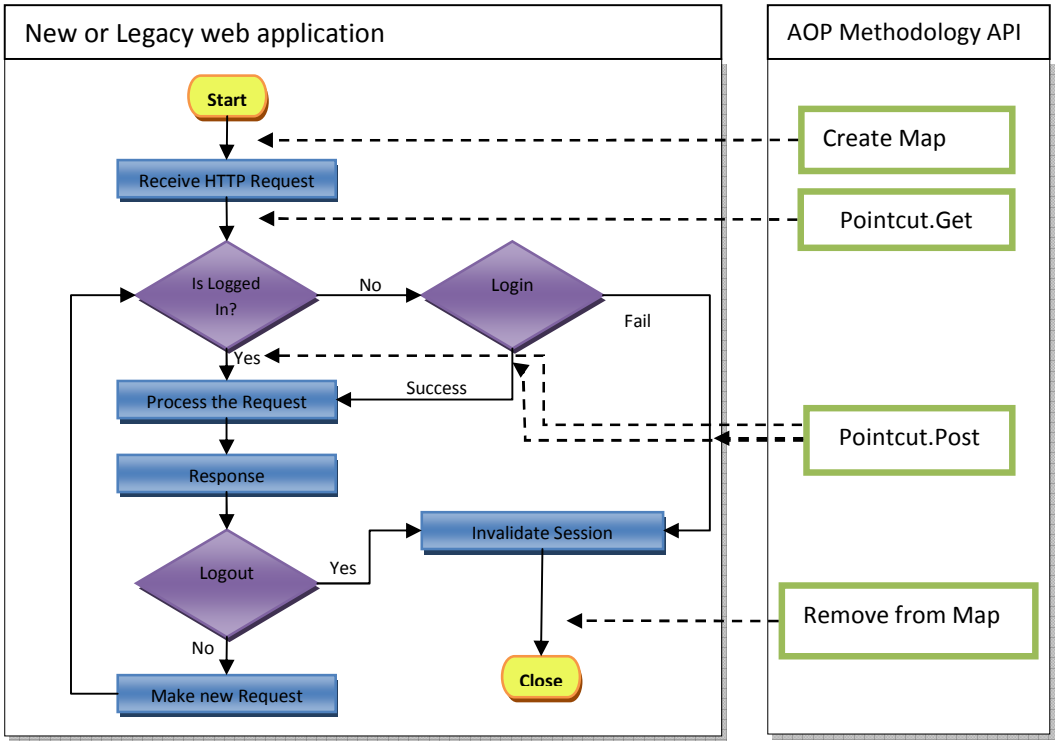

Figure 3.8 The Security Aspect flowcharts

After weaving, the above model (Figure 3.8) will be automatically translated (and more detailed) to the following model (Figure 3.9):

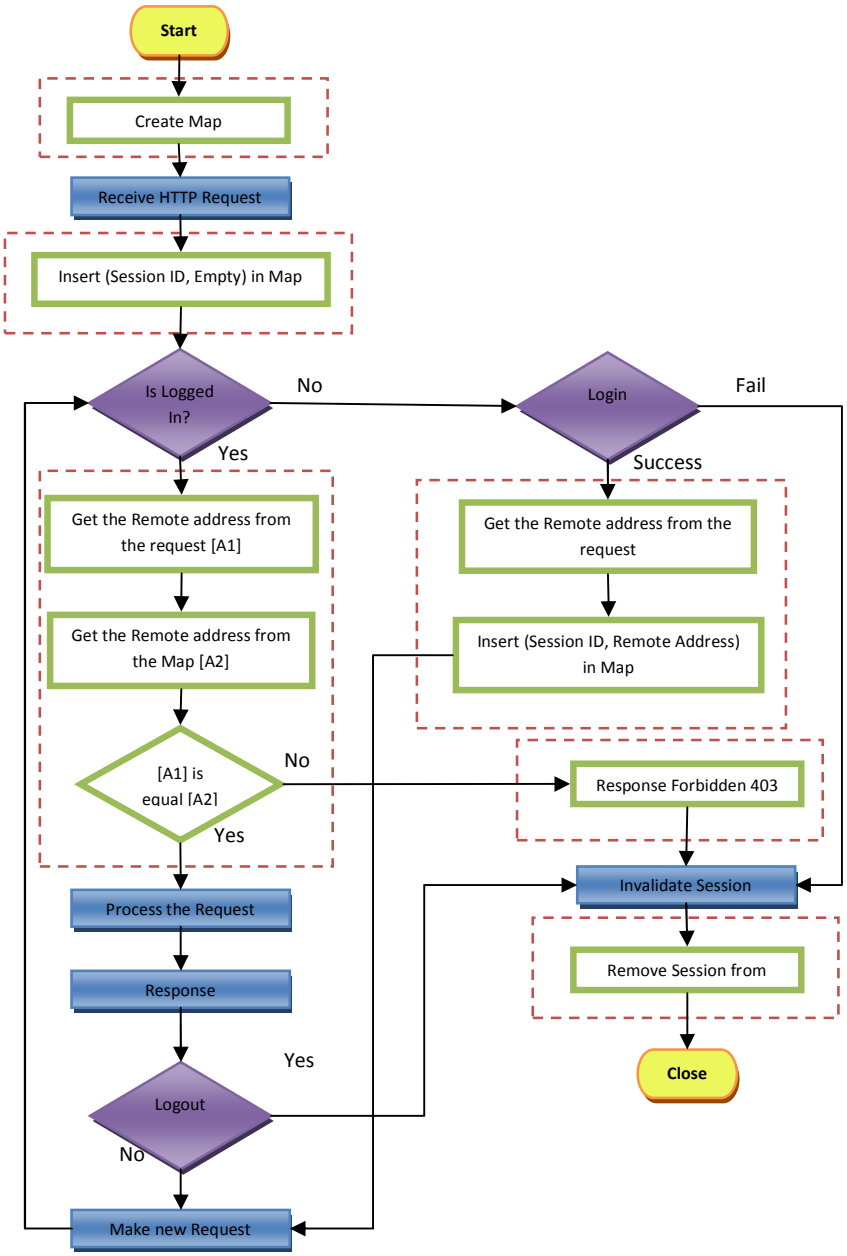

Figure 3.9 Application model after weaving

The application model after weaving done by the compiler the advice will appear as a part of the application, in the defined place, pointed by the pointcut.

3.4 A Case Study

In this thesis we will use a simple study case to better understanding. The case deals with university students getting their marks online via internet.

Study case scenario

We have a web application for a university web services for the students. Each student has a username and a password. With the user and password the students login in the server to ask for their marks. As response the marks are showed to student in the browser (Figure 3.10).

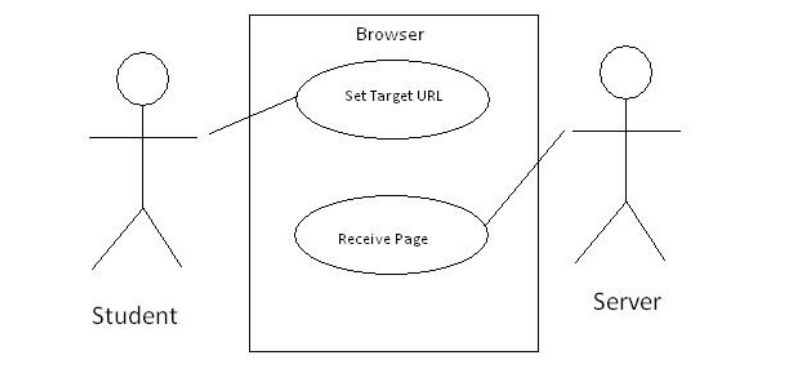

Figure 3.10 Use Case Diagram.

The student should login in the application to be authenticated. In the login process the password must match the one existed in the database. Once the login process is done successfully, the web server will set to the user a session id to identify him. During the session life time the web server will know that the coming request is from which user by checking the session id in the HTTP header parameter and do the appropriated response with the available requested data (Figure 3.11).

User Login

Ask for marks

Process marks

Show marks

Figure 3.11 Study Case Example System

The problem

The problem occurs in the session life time by a third part interception, besides the user and web server, which are the first and second parts in the communication, there is a third part (the attacker) who stole the session id for example using sniffing the network. This third part will have the valid session id, and he can send requests to the server using the session and successfully retrieve data like if he is the real user, without needing to know username and password. This problem is called broken Authentication and session management.

Apply the Solution

We will use the precede AOP methodology to enhance the security of the web application studying this case study above.

Adopting the Aspect-oriented programming model increases modularity by allowing the separation of cross-cutting concerns, forming a basis for Aspect-oriented software development. This Separation of concerns entails breaking down a program into distinct parts (so-called concerns, cohesive areas of functionality). Our concern is security, an HTTP request access security to the web server.

One view of AOP is the major feature of the program, core concern (business logic). In our simple study case we can say that the business logic is the university business. It involves services to the university students (student's marks). Another view is the cross-cutting concern (additional features), which is, in our case study, the checking from where the request is coming, independently from the business.

In our case we make security check advices that add a new authentication check for security on the user. The remote address advice is used to take the remote address of the user beside the session id. So an advice will check if the asked request is coming from the same remote address within a session life time.

To add the remote address criteria check to the application, we will use a hash map to associate the user session id to the user remote address of the place he used to login.

First the application starts and a Hash Map is created, the map is empty until users start to request the application.

Advices run in the specified join-points associated with each advice. In the case of a web application the join points are on the HTTP request process points. One of our points will set an initial mapping setting a raw for session id and an empty cell for the remote address. Also this point has an after execution body advice, executed after the method checking for user authentication (username and password).

This body advice checks around the associated remote address and session management.

- If the stored remote address is empty (it is the first time the user requests with the given session) then it associates the session id with the remote address.
- If the session is already associated with a remote address, and the current address (read from the current HTTP header) does not match the associated address in the map, Then this means that there is a problem and someone is trying to request data from the server using a valid session but from another place (another remote address), and it will get a proper advice response (in our study case it is a 403 HTTP response, the forbidden accesses page message).

The technology used in the case study application is J2EE, and the application is deployed to Tomcat web component connected throw JDBC with MySQL database (Figure 3.12).

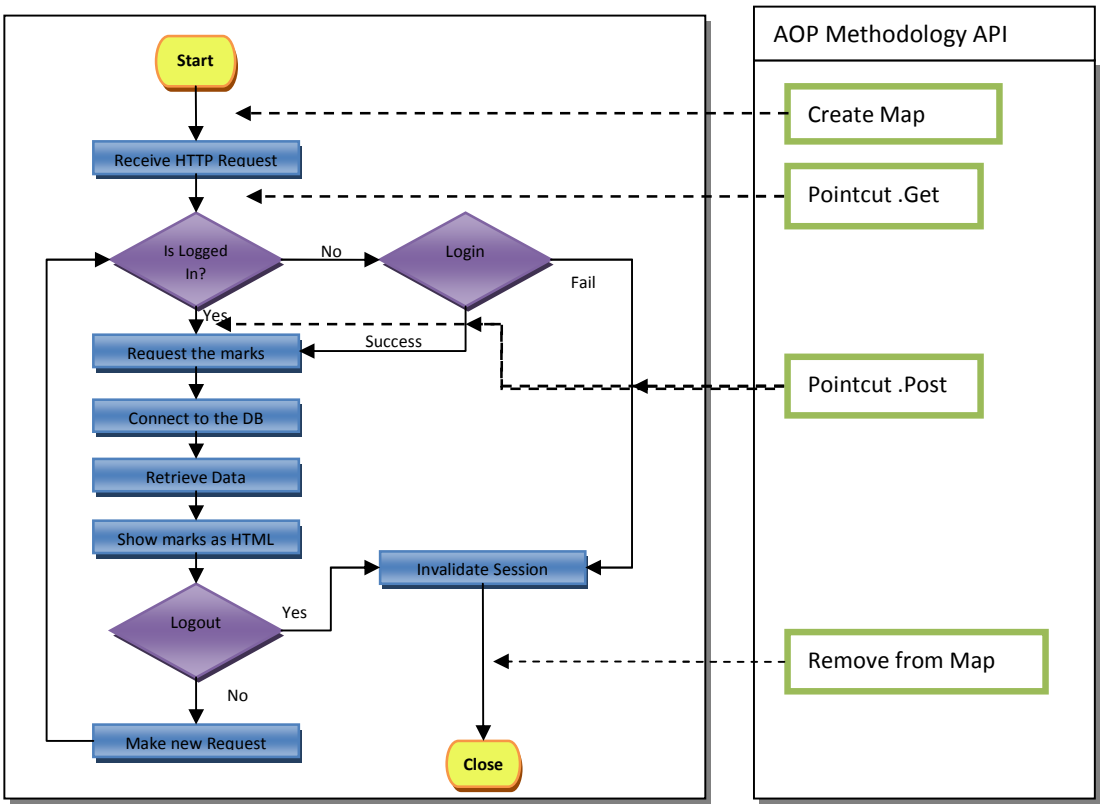

Figure 3.12 An example, using the Security Aspect API to secure the case study web application

The application is more secure after applied the Security Aspect API using the steps above (see the code in the Appendix).

3.5 Obtained Result

- The Obtained result is a secure checked web Application against session hijack, in broken authentication and session management. Supporting an application by a new aspect managing the session by adding a relation with the place the request comes from, identifying the real user. It increases the trust in user request, that the response data will be received by the real users.
  When the server receives a request from a stolen session id user, the server will be able to forbid this request, and the real user will stay able to request from the server normally.
- Security Maintenance: the used AOP methodology helps in the security maintenance. If there is any change needed, it can be done in a centralized point, the aspect advice.
- Legacy Application: The Security Aspect API can be applied to legacy web applications without changing in the application current code.

Chapter Four

Evaluation and Conclusion

4.1 Experimental Result
4.2 Thesis Evaluation
4.3 Thesis Conclusion
4.4 Future Works

4.1 Experimental Result

4.1.1 Experiment

With the study case web application running in the web server, a user with a username and password login in the application successfully and a session Id is assign to the user session.

The experiment has two parts,

4.1.2 Part one: The application is weak.

The session id is stolen it is the SessionId field in (Table 4.1), the stolen session Id is used to access the web server as the real user. A small ready java application was used to send the request with the stolen session Id. The result was as field Get success response from the server in (Table 4.1) shows there was "Yes" response from the server.

The experiment was done many times and the results are shown in the table below.

Table 4.1 Experiment results with the weak application

Session Id	Get success response from the server
A407ADD6C5CBB1A92FBDC1625D830AC1	Yes
3E24012829F49F325A0CC81E3905A810	Yes
140B3825FF02F00D7FC99E1473C365A2	Yes
937DD4F44DEAD063C2BA67CD2D0450BB	Yes
3C92F9C75DB907959F9C4FB76AE8E9C1	Yes
91405DA43D1C465F79416B9EAC0F9E72	Yes
588B046039A087C229FD6F6DF54ADF92	Yes
7073976B68F77D099F73EE37B6F8D583	Yes
BBB501C0D337BEB1944CAA6707604762	Yes
38F3C3E772893CBC039969946F90D5EB	Yes

4.1.3 Part Two: The same study case application applied the Aspect Oriented Paradigm Methodology:

Same steps were done again with the secure application: The session id is stolen it is the SessionId field in (Table 4.2), the stolen session Id is used to access the web server as the real user. A small ready java application was used to send the request with the stolen session Id. The result was as field Get success response from the server in (Table 4.2) shows there was "No" response from the server.

Table 4.2 Experiment results with the application applied the Aspect Oriented Paradigm Methodology

Session Id	Get success response from the server
6A758B36595DB9D3DBC188050B40407D	No
FC80981E8669A0E8C0F1DCF28FC7F692	No
4BF976E1E0857AE077E745A69CE2822B	No
465B4D1E2DA4B30E41B171201AE4DCA1	No
A593BA649F119889C52390DEF4304211	No
5AD5296D534B10B9AA2DA0340BC2487C	No
F710F46F09B27FED093871E4239F7940	No
ED6AD5AA75803E8D93FFAFD40F7BA7F6	No
DDC59A49FCB580B89B5203DC70881A90	No
762C6087DC571548FC86897C642F7085	No

4.1 Thesis Evaluation

For the broken authentication and session management problem, an already applied solution is the Secure Sockets Layer (SSL) which avoids man in the middle attack in the traffic between client and web server. So the client to server path should be encrypted.

The communication and credential storage has to be secure in transfer and storage. The SSL protocol for transmitting over HTTP confidential and personal data and documents should be the only option for authenticated parts of the application, and credentials should be stored in hashed or encrypted form.

The SSL can cause some problems, while most business hosting companies offer SSL or secure socket layer. For secure transactions from an online store, they may not be compatible with your trade account gateway. You will need to check with both your hosting company and your trade account to make sure that your gateway is supported before you select your hosting company. This is an area that cannot be fixed or worked around, so it is important to take the time to ensure compatibility ahead of time.

The SSL can be costly, because it will make an encryption decryption for all the data transferred so it will need more potential hardware and the encryption decryption data processes will take more time slowing down the application, and this point in web application is very important.

There is no proposed solution for the broken authentication and session management security problem using the AOP approach.

So the Advantage of our solution in this thesis by using the proposed AOP methodology is that is not expensive, applicable in all web applications without any heavy process that can slow down

the Web access. The AOP methodology is also not costly in the implementation; it can be used in the legacy web application.

4.2 Thesis Conclusion

This thesis has proposed an aspect-oriented programming approach to enhance web application security by solving the broken authentication and session management security problem. As a conclusion:

1. Enhancing the web security for web application, in the previous work of (Hemosillo et.al, 2007) they solved two problems on web application SQL Injection and XSS using AOP approach, in this thesis we contributed with an AOP approach solving another web security problem the Broken Authentication and session management. So we can conclude that the AOP is a suitable approach to solve web application security problems
2. Giving web application security concern a higher priority on the application design, without to change the business main importance.
3. Broken Authentication and session management depends on the HTTP communication between the server and the client and it depends also on the session lifetime.
4. Aspect-oriented programming can cover all the requests and response points in the application. Even when add new modules to the application.
5. In our thesis the aspect with the security code and the web application code is clearly separated. So there is no need for any modification in the web application existing code. Like this the aspect will be able to evolve independently.
6. The methodology proposed in the thesis is applicable in legacy and new web application.
7. Using the security aspect API in the web application the maintenance and advice changes is more centralized and easy.

4.3 Future Works

At our actual research state in this topic, we may state the following extensions as a future works:

- Solve more vulnerabilities using aspect-oriented programming as the Improper Error Handling.
- Study other characteristics in the client that can help with his identification.
- Propose a general methodology for AOP approach in web application security.

REFERENCES

Bostrom, G., March 2004. Database Encryption as an Aspect. Proceedings of AOSD 2004 Workshop on AOSD Technology for Application level Security (AOSDSEC).

Boudreau, T., Tulach, J., Wielenga, G., 2007.Rich Client Programming: Plugging into the NetBeans Platform. Copyright Sun Microsystems.

Cannings, R., Dwivedi, H., Lackey, Z., 2008. Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions. McGraw-Hill.

Chopra, V., Bakore, A., Eaves, J., Galbraith B., Li, S., Wiggers, C., 2004. Professional Apache Tomcat 5 . Wiley Publishing, Inc.

Chopra, V., Li, S., Genender, j., 2007. Professional Apache Tomcat 6 . Wiley Publishing, Inc.

Dave, G., Eric, P., Darren, J., 2006. Ajax in Action. Manning Publications Co.

Davis, S., Marrs, T., 2005. JBoss at Work: A Practical Guide. 1st Edition . O'Reilly

De Win, B., E., Joosen, Piessens, F., March 2003. AOSD & Security: A Practical Assessment. Workshop on Software Engineering Properties of Languages for Aspect Technologies (SPLAT) . AOSD'03. pp 16. Boston, USA.

De Win, B., Sanen, F., Truyen, E., Joosen, W., Südholt, M., July 2005. Study of the Security Concern. Network of Excellence on AspectOriented Software Development. Milestone 9.1.

Dittrich, D., April 1999. Anatomy of a Hijack. University of Washington, http://staff.washington.edu/dittrich/talks/qsm-sec/script.html

Eclipse Help , AspectJ Quick Reference

Garfinkel, S., November 2001. Web Security, Privacy & Commerce. 2nd Edition, O'Reilly,ISBN: 0-596-00045-6.

Griswold, W., Shonle, M., Sullivan, K., Song, Y., Tewari, N., Cai, Y., Rajan, H., February 2006. Modular Software Design with Crosscutting Interfaces. IEEE SOFTWARE

Hermosillo, G., Gomez, R., Seinturier, L., Duchien, L., December 2007. Using Aspect Programming to Secure Web Applications. Journal of Software, Vol. 2, No. 6.

Huang, M., Wang, C., Zhang, L., March 2004. Toward a Reusable and Generic Security Aspect Library. Proceedings of AOSD 2004 Workshop on AOSD Technology for Application level Security (AOSDSEC).

ISO/IEC JTC 1/SC 6 (Author), 1997. Information technology - Telecommunications and information exchange between systems - Use of OSI applications over the Internet Transmission Control Protocol (TCP) (Paperback). ISO/IEC 14766.

Jendrock, E., Ball, J., Carson, D., Evans, I., Fordin, S., Haase K., 2006. The Java EE 5 Tutorial. Published by Prentice Hall PTR.

Kawauchi, K., Masuhara H., March 2004. Dataflow Pointcut for Integrity Concerns. Proceedings of AOSD 2004 Workshop on AOSD Technology for Application level Security (AOSDSEC).

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G., 2001.An Overview of AspectJ. 15th European Conference on Object-Oriented Programming ECOOP, Budapest, Hungary.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J., June 1997. Aspect-Oriented Programming. Springer-Verlag. Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97). LNCS 1241. pp 220242.

Laddad, R., 2003. AspectJ in Action, Practical Aspect-Oriented Programming. Manning Publications Co.

Laddad, R., January 2002. I want my AOP Separate software concerns with aspect oriented programming. JavaWorld. http://www.javaworld.com/javaworld/jw012002/jw0118aspect.html.

Laney, R., van der Linden, J., Thomas, P., March 2004. Evolution of Aspects for Legacy System Security Concerns. Proceedings of AOSD 2004 Workshop on AOSD Technology for Application level Security (AOSDSEC).

Lieberherr, K., Lorenz, D.H., Ovlinger, J., September 2003. Aspectual Collaborations: Combining Modules and Aspects . Oxford University Press. Computer Journal, Vol. 46, No. 5, pp. 542-565(24).

Miles R., 2005. AspectJ Cookbook. O'Reilly Media, Inc.

O'Regan, G., January 2004. Introduction to Aspect-Oriented Programming. O'Reilly . http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html?page=2

OWASP, Accessed in November 2007. Ten Most Critical Web Application Security Vulnerabilities, http://www.owasp.org

Sadtler, C., Laursen, L., Phillips, M., Sjostrand, H., Smithson, M., Wan K., 2005. WebSphere Application Server V6: System Management and Configuration Handbook.

Stackowiak, R., Bales, D., Greenwald R., 2004. Oracle Application Server 10g Essentials. O'Reilly Media.

Stuttard, D., Pinto, M., 2008. The Web Application Hacker's Handbook. Wiley Publishing, Inc.

Verhanneman, T., Piessens, F., De Win, B., Joosen, W., March 2004. View Connectors for the Integration of Domain Specific Access Control. Proceedings of AOSD 2004 Workshop on AOSD Technology for Application level Security (AOSDSEC).

Viega, J., Bloch, J.T., Chandri, P., October 2001. Applying AspectOriented Programming to Security. Cutter IT Journal. Vol. 14, No. 2, pp. 3139.

Webhostinginformation.net, 2005. Disadvantages of Business Hosting, http://webhostinginformation.net/disadvantages-of-business-hosting/ .

Wong, C., 2000. HTTP Pocket Reference: Hypertext Transfer Protocol. O'Reilly.

Appendix A

ServletSecurityAspect.aj

```
package com.acme.aspects;

import java.io.IOException;

import java.util.HashMap;


import javax.servlet.ServletException;

import javax.servlet.ServletOutputStream;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


import com.acme.Util;

import com.acme.SessionTable;

public aspect ServletSecurityAspect {


        public pointcut getHTTPRequest(HttpServletRequest request,  HttpServletResponse response ) :

                        execution(* doGet (HttpServletRequest, HttpServletResponse))

                        &&

                        args(request, response);


        before(HttpServletRequest request, HttpServletResponse response )

        throws ServletException, IOException : getHTTPRequest(request, response )

{

                System.out.println("Aspect before the doGet method in LoginSerlet, getHTTPRequest
pointcut" );


                SessionTable.put(Util.KEY, Util.EMPTY);
```

```
                String sessionId = request.getSession().getId();
                System.out.println("Advice :sessionId = "+sessionId);
                String currentRemoteAddr = request.getRemoteAddr();
                System.out.println("Advice : " +
                                "currentRemoteAddr= "+currentRemoteAddr);
                System.out.println("Advice : " +
                                "Util.Key= "+Util.KEY);
                String storedRemoteAddr = (String) SessionTable.get(sessionId);
                System.out.println("Advice : " +
                                "storedRemoteAddr= "+storedRemoteAddr);
                if(storedRemoteAddr==null){
                        storedRemoteAddr ="";
                }
                if (storedRemoteAddr.equals(Util.EMPTY)) {
                        // this is the first-time request of the given session, associate
                        // the session id with the Ip address
                        SessionTable.put(sessionId, currentRemoteAddr);

                } else if (!storedRemoteAddr.equals(currentRemoteAddr)) {
                        // set the response to HTTP 403 (FORBIDDEN)
                        ((HttpServletResponse)
response).setStatus(HttpServletResponse.SC_FORBIDDEN);
                         response.sendRedirect("403.html");
                        //stop processing and return
                        return;
                }

        }
```

```
        after(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException : postHTTPRequest(request, response)
{

  System.out.println("Aspect after the doPost method in LoginSerlet with postHTTPReques" );
  // after empty




}


                public pointcut postHTTPRequest(HttpServletRequest request,
                                HttpServletResponse response ) :
                                execution(* doPost
                                                (HttpServletRequest,
                                HttpServletResponse))
                                &&
                                args(request, response);


                before(HttpServletRequest request, HttpServletResponse response )
                throws ServletException, IOException : getHTTPRequest(request, response )
        {
                        System.out.println("Aspect before the doPost method in LoginSerlet" );


                        SessionTable.put(Util.KEY, Util.EMPTY);
```

```
                    }
        after(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException : getHTTPRequest(request, response)
{

  System.out.println("Aspect after the doPost method in LoginSerlet" );



        String sessionId = request.getSession().getId();
        System.out.println("Advice :sessionId = "+sessionId);
        String currentRemoteAddr = request.getRemoteAddr();
        System.out.println("Advice : " +
                        "currentRemoteAddr= "+currentRemoteAddr);
        System.out.println("Advice : " +
                        "Util.Key= "+Util.KEY);
        String storedRemoteAddr = (String) SessionTable.get(sessionId);
        System.out.println("Advice : " +
                        "storedRemoteAddr= "+storedRemoteAddr);
        if(storedRemoteAddr==null){
                storedRemoteAddr ="";
        }
        if (storedRemoteAddr.equals(Util.EMPTY)) {
                // this is the first-time request of the given session, associate
                // the session id with the ip address
                SessionTable.put(sessionId, currentRemoteAddr);

        } else if (!storedRemoteAddr.equals(currentRemoteAddr)) {
```

```
                        //HIJACK!!! YOMMA! YOMMA!



                        // set the response to HTTP 403 (FORBIDDEN)

                        ((HttpServletResponse)
response).setStatus(HttpServletResponse.SC_FORBIDDEN);



                        //stop processing and return

                        return;

                }

                System.out.println("Aspect after the doPost method in LoginSerlet, End");

        }

                public static void main(String[] args) {

                        // TODO Auto-generated method stub



                }



        }
```

SessionTable.java

```
package com.acme;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class SessionTable {
        private static Map map = Collections.synchronizedMap(new HashMap());

        private SessionTable() {
        }

        public static void put(String key, String value) {
                System.out.println("SessionTable put");
                map.put(key, value);
        }

        public static void remove(String key) {
```

```
                System.out.println("SessionTable remove");
                map.remove(key);
        }

        public static String get(String key) {
                System.out.println("SessionTable get : key =" + key);

                return (String) map.get(key);
        }
}
```

Logout.java

```
package com.acme;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Logout {
 public static void logoutAction(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException{
            String sessionId = request.getSession().getId();
            SessionTable.remove(sessionId);
            request.getSession().invalidate();

    try{
        response.sendRedirect("index.jsp");
    }catch(IOException ioe){
        System.out.println (ioe);
    }
 }
}
```

LoginServlet.java

```
package com.acme.servlets;

import javax.naming.*;
import javax.servlet.http.*;
import javax.servlet.*;

import java.io.IOException;

import javax.sql.DataSource;

import com.acme.SessionTable;

import java.sql.*;
import java.util.ArrayList;
```

```
public class LoginServlet extends HttpServlet implements SingleThreadModel {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	private String connection = null;
	private String loginSQL = null;
	private String username = null;
	private String pass = null;
	private String loginErrorPage = null;
	private String loginSuccessPage = null;
	private String userRolesSQL = null;
	private int maxInactiveInterval = 1800;

	public void init(ServletConfig servletConfig) throws ServletException {
		connection = servletConfig.getInitParameter("connection");
		if (connection == null) {
			log("connection parameter is missing in LoginServlet");
			throw new ServletException(
					"connection parameter is missing in LoginServlet");
		}

		loginSQL = servletConfig.getInitParameter("loginSQL");
		if (loginSQL == null) {
			log("loginSQL parameter is missing in LoginServlet");
			throw new ServletException(
					"loginSQL parameter is missing in LoginServlet");
		}

		username = servletConfig.getInitParameter("username");
		if (username == null) {
			log("username parameter is missing in LoginServlet");
			throw new ServletException(
					"username parameter is missing in LoginServlet");
		}

		pass = servletConfig.getInitParameter("pass");
		if (pass == null) {
			log("pass parameter is missing in LoginServlet");
			throw new ServletException(
					"pass parameter is missing in LoginServlet");
		}

		loginErrorPage = servletConfig.getInitParameter("loginErrorPage");
		if (loginErrorPage == null) {
			log("loginErrorPage parameter is missing in LoginServlet");
			throw new ServletException(
					"loginErrorPage parameter is missing in LoginServlet");
		}

		loginSuccessPage = servletConfig.getInitParameter("loginSuccessPage");
		if (loginErrorPage == null) {
			log("loginSuccessPage parameter is missing in LoginServlet");
			throw new ServletException(
```

```
                                        "loginSuccessPage parameter is missing in LoginServlet");
                }

                userRolesSQL = servletConfig.getInitParameter("userRolesSQL");
                if (userRolesSQL == null) {
                        log("userRolesSQL parameter is missing in LoginServlet");
                        throw new ServletException(
                                        "userRolesSQL parameter is missing in LoginServlet");
                }

                String max = servletConfig.getInitParameter("maxInactiveInterval");
                if (max != null) {
                        try {
                                maxInactiveInterval = Integer.parseInt(max);
                        } catch (NumberFormatException nfe) {
                                log("maxInactiveInterval parameter value is non-integer, using the
defult value");
                                maxInactiveInterval = 1800;

                        }
                }
        }

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException {
                try {
                        response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED);
                } catch (IOException ioe) {
                        log(this.toString(), ioe);
                }
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException {
                boolean success = false;

                try {

                        HttpSession session = null;
                        session = request.getSession();

                        // long l= request.getSession().getLastAccessedTime() ;

                        String usernameValue = request.getParameter(username);
                        System.out.println("username : " + usernameValue);
                        if (usernameValue == null) {
                                response.getWriter().println(
                                                "<html><body>"
                                                                +                       "<h1>Missing
Parameter</h1><br><hr><br>"
                                                                + "Parameter <i>" + username
                                                                + "</i> is not found in the request
header."
                                                                +   "<br><hr><br></body></html>
");
                                response.flushBuffer();
```

```
                }

                String requestPassword = request.getParameter(pass);
                System.out.println("requestPassword : " + requestPassword);
                if (requestPassword == null) {
                        response.getWriter().println(
                                        "<html><body>"
                                                        +                       "<h1>Missing
Parameter</h1><br><hr><br>"
                                                        + "Parameter <i>" + pass
                                                        + "</i> is not found in the request
header."
                                                        +
"<br><hr><br></body></html>");
                        response.flushBuffer();
                }

                ArrayList userRolesList = new ArrayList();

                // String userRole=null;
                /*
                 * SQLManager sqlManager = SQLManager.getInstance(); Connection con
                 * = sqlManager.requestConnection(connection);
                 */
                Context initContext = new InitialContext();
                // Context envContext =
                // (Context)initContext.lookup("java:/comp/env");

                Context envContext = (Context) initContext.lookup("java:/comp/env");
                DataSource ds = (DataSource) envContext.lookup("jdbc/TestDB");
                Connection con = ds.getConnection();

                if (con != null)
                        System.out.println("con success!");
                else
                        System.out.println("Sorry , problems with con !");

                Statement stmt = con.createStatement();


                System.out.println("usernameValue" + usernameValue);
                // ResultSet rset =
                // stmt.executeQuery(loginSQL+" \'"+usernameValue+"\'");

                try {
                        ResultSet rset = stmt
                                        .executeQuery("select username, userpass  from users
where username = "
                                                        + usernameValue.trim());

                        if (rset.next()) {
                                String dbPassword = rset.getString(2).trim();
                                System.out.println("dbPassword: " + dbPassword);

                                System.out.println("requestPassword: " + requestPassword);
                                //
```

```
                                        if (dbPassword.equals(requestPassword.trim())) {
                                                success = true;
                                                // session = request.getSession(false);
                                                // if (session != null)
                                                // session.invalidate();
                                                // session = request.getSession(true);
                                                //
session.setMaxInactiveInterval(maxInactiveInterval);
                                                for (rset = stmt.executeQuery(String
                                                                .valueOf(String.valueOf((new
StringBuffer(

        String.valueOf(String

        .valueOf(userRolesSQL))))
                                                                                .append("
'").append(usernameValue)
                                                                                .append("'")));
rset.next(); userRolesList
                                                                .add(rset.getString(1)))
                                                        ;

                                                session.setAttribute("univ.role", userRolesList);
                                                session.setAttribute("univ.username",
usernameValue);

                                                }
                                        }
                                } catch (Exception e) {
                                        System.out.println(e);
                                }

                                stmt.close();
                                // sqlManager.returnConnection(con);
                                con.close();
                                // System.out.println(" loginSuccessPage -- "+loginSuccessPage);
                                if (success)
                                        response.sendRedirect(response
                                                        .encodeRedirectURL(loginSuccessPage));

                                else{
                                         SessionTable.remove(request.getSession().getId());
                                        response.sendRedirect(loginErrorPage);
                                }

                        } catch (Exception ex) {
                                log("LoginServlet", ex);
                                throw new ServletException(ex.getMessage());
                        }
        }

}
```

Success.java
package com.acme.servlets;

```
import java.io.IOException;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.acme.services.Marks;

/**
 * Servlet implementation class Success
 */
public class Success extends HttpServlet {
	private static final long serialVersionUID = 1L;

	/**
	 * @see HttpServlet#HttpServlet()
	 */
	public Success() {
		super();
		// TODO Auto-generated constructor stub
	}

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
	 *      response)
	 */
	protected void doGet(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {

		HttpSession session = null;
		session = request.getSession();

		System.out.println("remote address = " + request.getRemoteAddr());
		System.out.println("remote port = " + request.getRemotePort());
		System.out.println("headerNames = " + request.getHeaderNames());
		System.out.println("Session id  = " + request.getSession().getId());

		ServletOutputStream out = response.getOutputStream();
		out
				.println("<%@ page language=java contentType=text/html;
charset=windows-1256 "
						+ "    pageEncoding=windows-1256%>");

		out
				.println("<%@ page import=java.sql.*, javax.naming.*,
javax.sql.DataSource %>");
		out.println("<html>");
		out.println("<body>");
		out.println("<center>");
		out.println("<table cellpadding=0 cellspacing=0 width=100% border=1>");
		out.println("<tr>");
```

```
                out.println("<td >Online Services</td>");
                out.println("<td align=right><a href=logout.jsp>Logout</a></td>");
                out.println("</tr><tr>       <td width=30%>");
                out.println("<table width=100%>");
                out.println("<tr><td>Welcome </td></tr>");
                out.println("<tr><td><a href=Success?m=1 >Marks</a></td></tr>");
                out.println("<tr><td></td></tr>");
                out.println("</table>");

                out.println("</td><td width=70%>");

                if ((request.getParameter("m") == null)
                                || (request.getParameter("m").equals(""))) {
                        out.println("select an option");
                } else {
                        if (request.getParameter("m").equals("1")) {

                                //out.println("<%@ include file=marks.jsp %>");
                                Marks mrk = new Marks();
                                 String std = (String)session.getAttribute("univ.username");
                                try {
                                        mrk.printstdMarks(std, request, response);
                                } catch (SQLException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }

                        } else {
                                out.println("select an option");
                        }
                }
                out.println(" </td></tr></table></center>");
                out.println("</body>");
                out.println("</html>");
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
         *      response)
         */
        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub

        }

}
```