Journal of Basic and Applied Engineering Research
Print ISSN: 2350-0077; Online ISSN: 2350-0255; Volume 1, Number 8; October, 2014 pp. 50-54
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/jbaer.html

Client vs. Server Implementations of Mitigating XSS Security Threats on Web Applications

Prashant Singh[1], **P.N. Barwal**[2]

[1, 2]*Centre for Development of Advanced Computing, Noida, India*

***Abstract*— this paper begins by introducing the concept of Cross-site scripting (XSS) attacks. The introductory section gives brief information on web applications, web application standards and various types of security attacks. In order to examine the impact of XSS attack, this paper will talk about various types of Scripting attacks. The paper will then address the feasibility of various proposed solutions to mitigate this attack. This situation leads to further research regarding practical solutions in implementing a secure web application. This paper will also cover the new standards to improve the security of Web Applications. Then the paper looks for an optimized solution, measuring the pros and cons of client versus server implementations, for preventing XSS attacks without sacrificing the usability of the web application. Finally, this paper ends with the conclusion of compared results, highlighted issues and solutions.**

Keywords: **Web Application Standards, Security Issues, Security Measures, XSS Scripting**

1. INTRODUCTION TO WEB APPLICATIONS

Web yapplication is a software that runs on the web i.e. using a web browser. These applications can be made using any technology and with the help of languages supported by web browsers. Over the past decade or so, web has become a huge source of data and a lot of web applications are used for connecting and sharing information with its users. From a technical point of view, web is an environment that allows a variety of customizations and also allows to deploy applications built using different technologies on various platforms. These applications can be made available to millions of users at a very fast and inexpensive rate. This has become the reason for the exponentially increasing popularity of these applications. So the basic architecture of a web application consists of an environment – web, tools to build the application – programming languages and the interface to run the applications – web browsers.

"World Wide Web Consortium (W3C) is an international Community that develops open standards to ensure long-term growth of the web". [1] It is responsible for setting up standards for the various technologies used in web to make it suitable and secure for the common user. It also helps in educating people about web. It develops protocols and guidelines to ensure the growth of web. It defines standards for the following technologies under Web Design and Applications.

HTML & CSS, JavaScript Web APIs, Graphics, Audio and Video, Accessibility, Internationalization, Mobile Web, Privacy and Math on Web.

With the increase in the amount of data present in web, it has become indispensable to take care of the security of the web applications that retrieve this data from the databases and present them to the client. Since the information accessed by web applications is very sensitive and can be used by any malicious body for their benefits, it's the need of the hour that we secure them. Web applications have evolved drastically, and so have the ways to exploit them. Few of the attacks that are known are Cross site scripting, SQL Injection, Path Disclosure, Denial of Service, Code Execution, Memory Corruption, Cross site request forgery, Information disclosure, Arbitrary file, Local File include, Remote File include, Buffer Overflow etc.

In this paper, we talk about the XSS attacks commonly known as Cross-Site scripting attacks. This attack is a major threat to most of the web applications as it may result in a severe loss of sensitive data. There are various methods to avoid this attack, we'll be talking about the client as well as server side implementations of mitigating this attack. And in the end we'll try to come up with simple steps that can be taken at the server / client end in order to avoid this attack at the very basic step.

This paper is outlined as follows. *Section I* provides the introduction to Web application architecture and also covers its basic components and common standards for developing web applications. *Section II* describes various security threats of web applications. This section covers most of the common attacks in brief and describes Cross Site Scripting in details. *Section III* describes the various code-level implementations to avoid Cross Site scripting attack. This section covers the server and client side implementations of mitigating this attack. *Section IV* provides some of the practical solutions for securing web applications. This section addresses some important guidelines for the users (Application administrators) to secure their Web application at the code level. *Section V* highlights a few research topics that can be further explored to strengthen the security of these web applications.

Study of securely using web application against other attacks, such as SQL injection etc., is outside the scope of this paper.

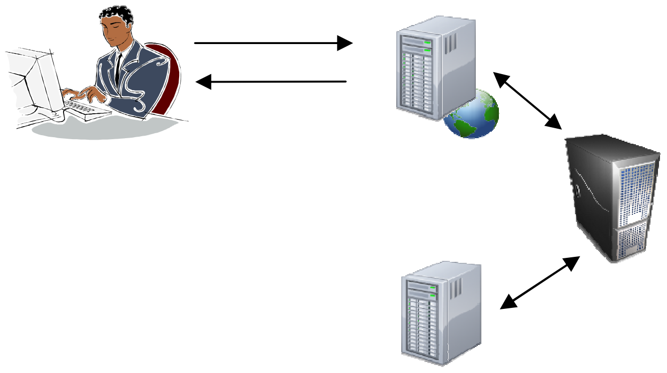
1.1 WEB APPLICATION

One important advantage of Web application ease of access for a huge amount of users. Installing a web application is easy and eliminates the need to provide multiple revisions as in case of desktop based applications. Broadly a web application comprises of Web components, a database and a client to render the web application – Web Browser.

1.1.1 Web components

Web Components can be web servers, servlets, web pages, web service endpoints, or JSP pages.

1.1.2 Architecture



1-1 Web Application Architecture

Client sends the request to the web servers, which then process the request and fetch the required data and return back to the client.

1.1.3 Security Issues [2]

Despite their advantages, these applications may result in a lot threats to the integrity and security of the data they linger on. The main reason that web applications get compromised easily are that:

- They need to be available to public all the time to provide service to its users.
- Firewall and SSL are not sufficient to provide protection against hacking of web applications, as access to these web applications has to public.
- These applications have direct access to the backend data, and thus require additional level of security.
- Most of the web applications are custom made, and hence are not tested well for all kinds of malicious attacks.

1.2 CROSS SITE SCRIPTING

Cross-Site scripting refers to the technique of compromising a web application by sending a malicious piece of code to the web application. It helps the attacker to collect data from the genuine user or may result in unauthorized access/use of the web application database. It is considered as the most common attack in web applications. [3]

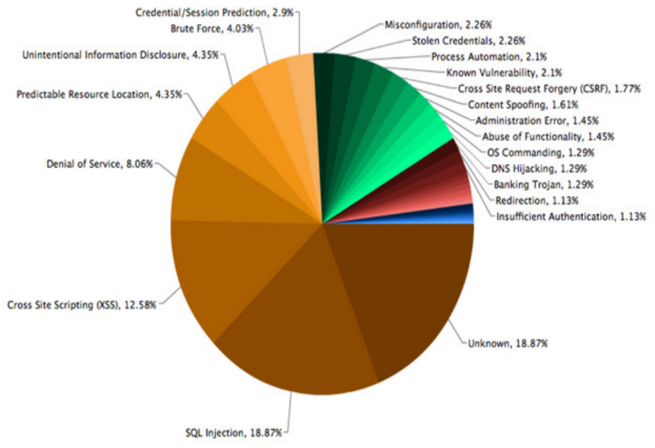


Fig. 1-2 Web Hacking Incident Database 2011 (WHID)

The pie-chart clearly shows that the extent of SQL Injection followed by Cross site scripting is maximum amongst the rest of the hacking incidents. Web applications are made up of either static web pages or dynamic web pages. Static web pages contain fixed information and the content of these pages can be trusted at all times. Whereas, dynamic web pages are generated with the content taken at various steps, the attacker, at this step embeds the malicious code (in a browser understandable language) in the web application dynamic page. This malicious code is then rendered by the web browser of the client and thus this attack takes place.

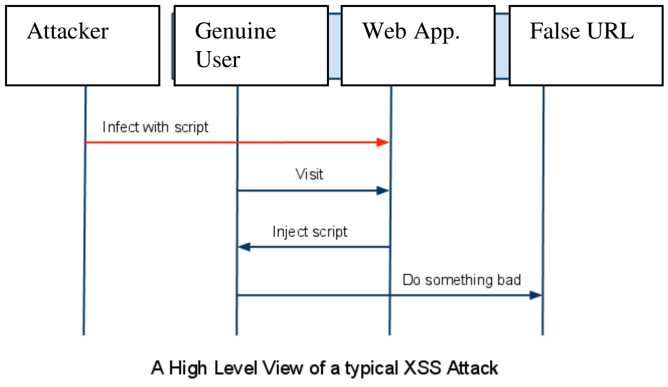


Fig. 1. 3 XSS attack [3]

Figure 1.3 shows the high level diagram of how a XSS attack works. Firstly the malicious script is used by the hacker to infect the original web application. It is injected in the system in the form of input to the web application. The real user, when uses the web application, falls prey to the script injected by the attacker. And the script results in either sacrificing the data of the genuine user, or it results in taking the user from the web application to some other place where the attacked can take advantage of the user data. Cross site scripting can be classified broadly into two categories:

1.2.1 Non-Persistent or Reflective

In this type of XSS attack, the attacker sends the malicious URL to the genuine user by the means of email or message. If the user clicks on the URL, a request is sent from his browser to the web application and the malicious code is injected in the user's browser. This script is then automatically executed, to obtain information regarding the client. The malicious script thus stays on the client browser. For example, if an attacker injected a script in the browser, accessing any google sites while logged in will transfer the information to the attacker without the notice of the genuine user.

1.2.2 Persistent or Stored

In Persistent XSS attack, the malicious code is saved at the 'trusted' server. Attacker inputs the malicious piece of code as a part of the genuine information and the information gets stored at the local database of the web application permanently. So, whenever any user requests that information, the malicious output is returned to them. For example, a simple application that takes name of the user as input. Displays the link of the user name and when clicked, redirects to the user profile. Attacker will inject the malicious script along with his / her name and when clicked, it will redirect to whichever place they want. This type of attack is more severe than the Non-Persistent version as the genuine information gets modified for all the users by attacking only once.

1.2.3 DOM Based XSS attacks

Persistent and Non Persistent XSS attacks mention the transfer of malicious information from client to server or server to client. This type of XSS attack occurs when the malicious information is injected in the client browser and the information stays in the client itself. As defined by Amilt et Al [4], "The entire tainted information flow from sink to source takes place in the browser"

The broader level of categorization of XSS attacks would be as Server XSS and Client XSS.[5]

1.2.4 Server XSS

This category of XSS attacks include those where the malicious script injected by the attacker becomes a part of the response generated by the web application server. The source of the data could be from a request or from a stored location, so both Persistent and Non-Persistent types fall under this category.

1.2.5 Client XSS

It occurs when the malicious data is used to update the DOM with a malicious JavaScript call. The source of the data could be from the DOM or from the server. But the ultimate source of data is either the request from the client or stored at the server. Thus, Persistent and Non-Persistent types also fall under this category. DOM based XSS attacks are a subset of this type.

1.3 SECURITY MEASURES

As discussed in the previous sections, XSS attacks pose a server threat to the web application and its users. So we need to find ways to mitigate this threat. Broadly there can be two categories for avoiding XSS threats:

1.3.1 Client-Side implementations:

The basic aim of these implementations is to validate the input coming from the client. We will identify a few client side XSS mitigating solutions.

1.3.2 Noxes tool [6]

It is a Microsoft-Windows based tool that acts a personal firewall and runs as a background service on the system of a user. Noxes provides an additional layer of protection as compared to the traditional firewalls placed in our systems. It allows users to create filter rules for the web requests. Noxes is a promising client side tool that can help avoid the XSS attacks and also its advanced versions, but it still requires the user to cancel the operation that would lead to an attack. As mentioned in their research, the target users for this tool are people with a certain level of "technical sophistication". Few limitations of using Noxes are:

1. It is not a freeware. Web applications that require support of an open-source / freeware will not opt for such solution.

2. It is not an automated solution that works for all the websites. User has to manually define the rules, like in case of firewalls for each website.

3. It lacks the SSL support.

1.3.1.2 ntiSamy [7]:

AntiSamy is an Open Source web input validation encoding project by OWASP (Online Web Application Security Project). It also has the option of defining rules as a part of its

XML configuration file, where the user can define the types of inputs allowed for a particular web application. Currently it is released only to support web applications developed using Java and .Net technologies. The input HTML is supplied to the Antisamy filter where all the input parameters are validated against the policies defined in the XML configuration file. If the input violates the policy, a user friendly message is shown and the input is removed from being inserted in database.

The workflow of AntiSamy project is as follows:

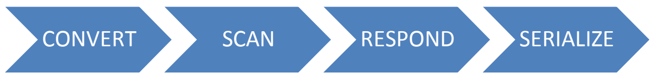


Fig. 2 1 Workflow of AntiSamy Project

Firstly, the input from the user is converted to Parser understandable format. It allows the creation of DOM objects and provides clean output. It also prevents the fragmentation attacks. The converted file is then scanned and the policy file is used to define responses for each of the tags present in the converted file. Then each tag is validated and malicious content is filtered while creating the response. Then finally the converted file is serialized back as HTML/XHTML.

The disadvantages of using this technique are:

1. Currently the support is only for Java and .Net frameworks.
2. The payload increases, as the processing time between the request and server response gets increased by adding the additional layer in between.

1.3.2 Server side Implementations:

Server side implementations allow attacker to inject the malicious code at the client end, and filter them out at response time.

1.3.2.1 DeDacota [8]:

Doupé, Adam, et al propose deDacota, a tool to prevent server side XSS attacks. It is based on the technique that the JavaScript is separated from the HTML code of the web application. The semantics of the web application are kept intact and the code is then re-written, free of malicious scripts. DeDacota is basically the prototype tool built in ASP .Net framework. It also acts as a static engine analyzer. It works on the binaries of the web applications to statically sanitize them.

The limitations of using deDacota kind of prototype in a real-world scenario are:

1. Applications that are built in ASP.Net use the dynamic language features of ASP .Net to decide the output of the application will not have a detailed static analysis of the web application.
2. It does not support complex string operations, like regular expressions.
3. The prototype – deDacota, does not handle JavaScript in the HTML attributes.

1.3.2.1 XSS Filter:

In other implementations, Duraisamy A, et al talk about filtering the web application inputs at the server end. They use a JavaScript detection algorithm to identify scripts in the input. Then they use a reverse proxy component to extract the request parameters and return the parameter to the JavaScript analyzer. Then the XSS filter sanitizes the malicious scripts to filter out clean code. Then comments are removed, so as to balance HTML tags, it also removes blank spaces etc. It finally uses a Data Access object to fetch data instead of the type and implementation of actual data. It helps in moving data source to a different location without having to change the business logic.[9]

The main limitations of using this kind of approach are in terms of the overhead filtering process creates, as it is a multi-staged filtering mechanism. The payload of the application increases and this in-turn arises the question of usable security.

1.4 GENERIC SOLUTIONS

The various steps that can be taken at the application architecture level are:

1. Input validation or Input Refining – All the inputs to the web application must be carefully monitored, and only white listed inputs should be allowed in the application.
2. Escaping essential HTML characters – While implementing any XSS filter, it is important to whitelist desired characters carefully. For example, &lt and &gt characters may be required in the HTML, but they attract filters' suspicion as malicious characters.
3. Using HTTP Only Cookies only – Using HttpOnly flag while creating cookies helps protect the client side script accessing the protected cookie.
4. Escape, JavaScript, CSS, attributes and URLs before input in the system.
5. Using CSP (Content Security Policy) to secure the whole web application – It's a W3C specification that tells informs the client web browser about the location and the type of the resources that are whitelisted.

1.5 ANALYSIS OF TECHNIQUES

1.5.1 Client Side Implementations:

Pros and Cons of the client side implementations:

Pros:

1. *Filtering the data at the client end* - Client side implementation as discussed, filter the input fields before sending the request to the database. Thus, it prevents the minimal chances of sending the malicious code to the database server.
2. *Compatibility*: Since these filters are enabled at the client end, they don't rely on technology or framework of the application that is being made. These implementations irrespective of the technology are able to filter out the malicious code.
3. *Database Friendly:* Since the malicious code is removed at the client end only, the database remains unaffected of the attack.

Cons

1. *Overhead :* Since all the filtering is done at the client end, the overhead before sending the data to the server increases, thus the overall time for sending the request to the server increases and hence the response time of the web application gets delayed. This is a major issue as the web applications are designed to be prompt and real-time.
2. *Extensive White-List Rules:* It requires to have a detailed white list of characters that can be allowed in the database. Once the XSS filter is surpassed, there is no additional security for existing server data. Also, a restricted list would result in string-matching errors.

1.5.2 Server side Implementations

Pros and Cons of the server side implementations

Pros:

1. *Less Initial Overhead*: Since server side implementations allow user to send the request without any filtering whatsoever, the response time of the server does not take a hit. Filtering is carried out as a separate thread in the background and filtered responses are returned.
2. *Existing data sanitization:* The mechanism of server side filtering involves separation of JavaScript from the HTML code, and hence the existing code can also be sanitized apart from the live incoming inputs.

Cons:

1. *Database Overhead:* Since this approach allows user to input any data, genuine or malicious, the amount of data present in the database may grow rapidly. Thus increasing the efforts required to filter out malicious code.
2. *String-matching / Double Injection-* Server side implementations are prone to string matching issues as the request parameters sent by the browser may be stored differently at the server side (encoded). Also Double Injection where a concatenated malicious code is sent also needs to be flagged off.

2. CONCLUSION

The general idea of Web application, standards, and XSS attacks was basically to provide a solution that can be incorporated to avoid this attack effectively without the loss of usability of web application. Use of web applications has evolved and is still currently evolving very rapidly towards offering fast services with the notion of usable security. We describe a few of the client vs server side implementations of mitigating these attacks. However, the need of the hour is to make a solution that can be incorporated at both the ends, without overloading the application. These loop holes have to be covered at the very basic architecture level of the application. Then only a client side or server side protection will be useful. These problems can be solved with some new standards, such as the CSP proposed by W3C. For the time being, Web application vendors and developers can protect their application by using common filters like the AntiSamy Filter plus by providing an additional server filtering layer. Where the response from the server can also be filtered in real-time using similar policies as used by AntiSamy.

REFERENCES

[1] "World Wide Web Consortium (W3C)" *http://www.w3.org/*

[2] "Web Applications: What are they? What of Them?" *http://www.acunetix.com/websitesecurity/web-applications/*

[3] "Cross Site Scripting Attack": *https://www.acunetix.com/websitesecurity/cross-site-scripting/*

[4] "DOM Based Cross Site Scripting or XSS of the Third Kind" (WASC writeup), Amit Klein, July 2005 *http://www.webappsec.org/projects/articles/071105.shtml*

[5] 'Types of Cross – Site Scripting" *https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting*

[6] Kirda, Engin, et al. "Noxes: a client-side solution for mitigating cross-site scripting attacks." *Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006.*

[7] "OWASP AntiSamy Project" *https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project*

[8] Doupé, Adam, et al. "deDacota: Toward preventing server-side XSS via automatic code and data separation." *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013.*

[9] Duraisamy, A., M. Sathiyamoorthy, and S. Chandrasekar. "A Server Side Solution for Protection of Web Applications from Cross-Site Scripting Attacks."*International Journal of Innovative Technology and Exploring Engineering, ISSN: 2278-3075.*