International Journal of Computer Applications Technology and Research
Volume 3– Issue 10, 604 - 609, 2014, ISSN: 2319–8656

Study of Cross-Site Scripting Attacks and Their Countermeasures

Gurvinder Kaur
University Institute of Engineering and Technology
Kurukshetra University
India

Abstract: - In present-day time, most of the associations are making use of web services for improved services to their clients. With the upturn in count of web users, there is a considerable hike in the web attacks. Thus, security becomes the dominant matter in web applications. The disparate kind of vulnerabilities resulted in the disparate types of attacks. The attackers may take benefit of these vulnerabilities and can misuse the data in the database. Study indicates that more than 80% of the web applications are vulnerable to cross-site scripting (XSS) attacks. XSS is one of the fatal attacks & it has been practiced over the maximum number of well-known search engines and social sites. In this paper, we have considered XSS attacks, its types and different methods employed to resist these attacks with their corresponding limitations. Additionally, we have discussed the proposed approach for countering XSS attack and how this approach is superior to others.
Keywords: - Cross-Site Scripting (XSS), Malicious Injection, Web Security, and Web Application Attacks.

1. INTRODUCTION

With the everywhere-ness of information superhighway, i.e. Internet, organizations are serving people with their business on web. However, as the owners of the business emphasize greater on their business logic they do not get concerned about the vulnerabilities and security hazards inclined to their websites. Web Security describes the guidelines used to block threats to diminish the web attacks. An attack may be feasible due to the existence of vary types of flaws and bugs in the coding. As per Ponemon Institute Life Threat Intelligence Impact Report 2013 if the actionable intelligence about cyber attacks is available only 60 seconds before then the average cost of exploit could be reduced to 40 percent [1]. That is if we have an appropriate method to handle an attack at the very first step then the cost of the damage caused due to that attack can be diminished largely.

The inaccurate authorization and sanitization of data given by web server has brought in the accountability for XSS attacks. It is the attack on the secrecy of customer of a specific website by approving injection of inputs containing HTML tags and JavaScript code. As per OWASP (Open Web Application Security Project) 2013 release cross-site scripting is one of the major attacks performed [2]. Cenzic Application Vulnerability Trends Report 2013 confers that among the top 10 attacks 26% comprises of XSS attacks only [3].

The rest of the paper is organized as follows:

Section II discusses the web application architecture. Section III discusses the XSS attacks and its types in detail. Section IV provides the survey explored with their relative weaknesses .Section V discusses the proposed approach and how it would be better. Finally, section VI concludes the paper.

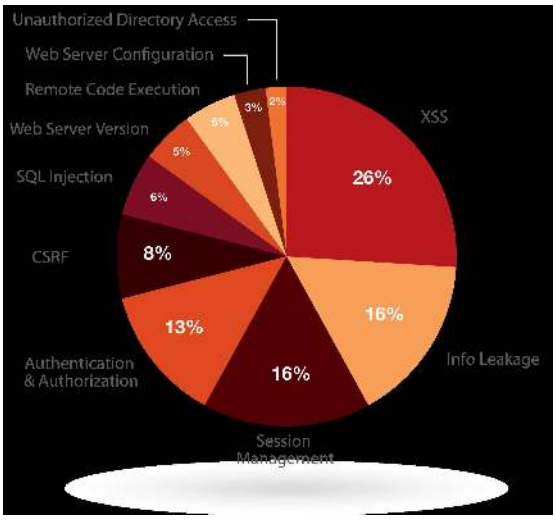

Fig1: Web Application Security Vulnerability Population [3]

2. WEB APPLICATION ARCHITECTURE

As XSS, attacks occur over the application layer so it is important to know how the web application works over the internet. Web Application has three-layered architecture as shown in Fig 2.

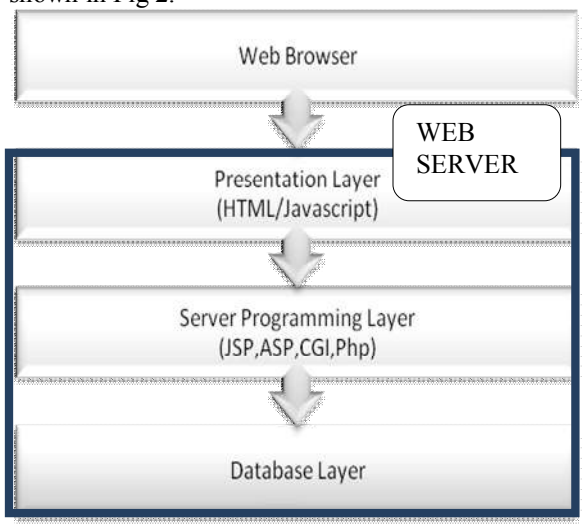

Fig 2: Web Application Architecture [4]

1.) *Presentation Layer:* This tier accepts input from end user and display output to user. It functions as a graphical user interface (GUI). It literally attach with the client.
2.) *Server Programming Layer:* This layer is located between presentation layer and database layer. Data processing is handled in this layer and it can be programmed in any of server scripting languages like JSP, PHP and ASP etc.
3.) *Database Layer:* This tier stores and manages all the delicate data of web application. This layer is responsible for access of authenticated users and rejection of malicious users.

3. CROSS-SITE SCRIPTING ATTACKS

In this section, XSS attacks & its types are discussed in detail. An XSS attack is one of the most common web application attacks that are used by hackers to sneak into web applications. In XSS, attacker embeds malicious script into a website. Whenever a user browser run this code the attacker can shape the browser to do whatever it wants .XSS attacks occur whenever an application takes un-trusted data and sends it to web browser without proper validation and sanitization[5]. So in XSS attacks three parties are involved- the attacker, the client and the website. In XSS attacks, the attacker insert malicious scripts to target websites for session hijacking, cookie stealing, and malicious redirection. This attack arises, as the web server does not appropriately assure that generated pages are properly encoded to avert the inadvertent execution of scripts and when input is not justified to prevent malicious HTML from being displayed to the users.

Example**:**
<% out.println(" welcome " + request.getParameter("name")); %> **(Example of poorly –written code on Web server- saving it as test.jsp)**

Case 1: Normal User

```
<HTML>
<BODY>
Welcome Stefan
</BODY>
</HTML>
```

Output:

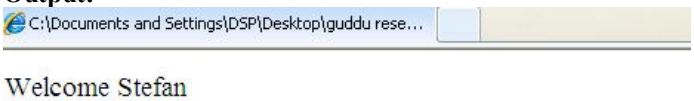

Fig 3: Response of a normal HTML code

Case2: Attacker

```
<HTML>
<BODY>
Welcome <script>alert ("Attacked") </script>
</BODY>
</HTML>
```

Fig 4: Output after code inserted by attacker

The example above shows a web site having XSS vulnerability. In Fig 3 a normal user when enters his name it is displayed. In Fig 4, an attacker misleads the application server by entering the JavaScript in place of name. This is quite a simple example.

There are primarily three types of cross-site scripting attacks as follows:

1. *Stored or Persistent attacks*- This attack appears when the malevolent code is sent to a web application where they are stored permanently on the mark server. When a user requests for the stored information this script is executed in its context. Examples for raider favorite targets include web mail messages, forum, comment field, visitor log etc. this attack occurs in the following manner:

-Malicious script is inserted into a web form by the attacker, which is then stored by the server in the database.

-When a request is send to the infected page by the user, the script is executed in the user's browser and the user's cookies are passed to the attacker.

2. *Reflected or Non – persistent attacks*- In this the malicious code is breed into a URL as a value of parameter, as a HTTP query in such a way that the rebounded content consists of unprocessed script. This attack occurs in a following way:

-Attacker examines a website and finds vulnerability on a web page.

-Attacker may embed a URL to exploit the weakness and may send an email to the user captivating the user to click on a link for the URL under false charade

-The URL will point to the normal website, but it will contain attacker's malicious code that the site will reflect

-User will visit the URL provided by the attacker while log into the given website

-The malicious script is executed on the user's browser as if it came directly from the web server.

-By this way, an attacker can steal sensitive information through user's cookies [6].

3. *DOM (Document Object Model)-based XSS attacks*- This exists within a site and can be used in a reflected manner. In this case, the malicious data exists solely in the browser and is not sent to the server. A brief example of a DOM-based XSS attacks would be a modifying the web history of a user.

4. RELATED WORK

Johns *et al.* has proposed a passive detection system to identify successful XSS attacks [5]. It uses two different approaches based on generic observations of XSS attacks and web applications. In this reflected attack is detected by a request/response matching which is based on the direct relationship between the input data and the injected scripts. In this the input parameters and the scripts found in final HTML is converted into a non-ambiguous representation by removing all encodings and the appropriate matching is done by constructing a DFA for each of the input parameter. For stored attacks, it adopts a generic XSS detection using a list of known scripts in which they used a training based XSS detector in which list of all outgoing script is matched up with the detector's known list. The weakness of this system is that it uses different implementation schemes for the two types of XSS that increases the overhead. It just detects the already existing attacks and false positives are there.

A static analysis for finding XSS vulnerabilities is demonstrated by Wassermann & Su [7] that straightforwardly addresses weak or absent input validation. The approach integrates work on tainted information flow with string analysis. The proposal has two parts: (1) an adapted string analysis to track untrusted substring values, and (2) a check for untrusted scripts based on formal language techniques. .String-taint analysis not only represents the set of string values a program may create, it also defines the formal language representation with labels that indicate which substrings come from untrusted sources. The second phase of the method enforces the policy that generated web pages include no untrusted scripts. It has many disadvantages like the tool produces false positives and it failed to resolve certain alias relationships between variables whose values are used for dynamic features. It failed to detect the DOM-based XSS. The string analysis-based tool could not handle arbitrarily complex and dynamic code.

Wurzinger *et al.* [8] introduced a tool known as SWAP (**S**ecure **W**eb **A**pplication **P**roxy), a server-side solution for discovering and preventing cross-site scripting attacks. SWAP contain a reverse proxy that intercepts all HTML responses, as well as a make use of modified Web browser to detect script content. SWAP contains a JavaScript detection component,

which is able to determine whether script content is present or not, a reverse proxy, which block all HTML responses from the server and subjects them to analysis by the JavaScript detection component and a set of scripts to axiomatically encode/decode scripts/ script IDs. SWAP introduces a performance overhead. It cannot guard counter to other kinds of objectionable content, such as static links pointing to sites including malicious scripts

In this paper a result for unit testing and action-level security of Struts Web applications is demonstrated by Wu *et al.* [9] by experimenting the applications from model-view-controller (MVC) respectively, and safe Struts applications with the help of different access control implementations. The aim of using struts is to neatly separate the model (application logic that communicates with a database) from the view (HTML pages shown to the client) and the controller (instance that passes info between view and model). JUnit tests the model, and StrutsTestCase does the testing of controller, while HttpUnit does the testing of view. The action level security solution comprises of four stages: access control in actions and JSPs, by extending the request processor, access control by servlet filtering, & WEB-INF. The solution of Struts Web application unit testing and action-level security just extended the general Web unit testing methods. The performance of application is degraded.

Galan *et al.* [10] suggested a multi-agent system for the automated scanning of web sites to disclose the existence of XSS vulnerabilities exploitable by a stored-XSS attack. The set of agent's part of the proposed architecture and the operation of the scanner are as follows. A webpage parser agent crawl the web application from which information about the different web forms found is used to build a repository of potential injection points (Injection point repository). A script injector agent reads the list of injection points recognized by the parser agent and also makes a selection of vectors attacks from the Attack vector repository. The desired set of attack vectors is launched against each of the potential points of attack of the application. A list of the performed attacks is stored in a Performed attack list. The verificator agent gets the list of the attacks to be verified and crawl the web application looking for each of the attacks. A report about the results of the scanning process is elaborated and stored. This approach has not better performance & accuracy.

The approach by Putthacharoen & Bunyatnoparat [11] aims to change the cookies so that they would become impractical for XSS attacks. This technique is called "Dynamic Cookie Rewriting" enforced in a web proxy where it will automatically put in place of the cookies with the randomized value before sending the cookie to the browser. In this way browser will keep the randomized value rather of original value sent by the web server. At the web server, end the return cookie from the browser over rewritten to its original form at the web proxy before being dispatched to the web server. So in case if XSS attacks swipe the cookies from the browser's database, the cookies cannot be used by the attacker to imitate the users. Four domains are kept to identify the cookie that is Name, Domain, Path, and Port. The initial value of the cookie and the randomized value are also kept in the same table. This table is reserved at the web proxy database server. The web proxy server will use this information to rewrite back cookies. The drawback of this approach is the compatibility problem that occurs while implementing the proxy server and the single point failure issue

In [12] a technique is proposed which is invoked when user injects code in the field of web application by V. & Selvekumar. .It uses the complete HTML parser and JavaScript Tester to detect the presence of JavaScript for filtering it out. The user created HTML content is passed to the XSS sanitizer and the static tags are checked. The static tags are retained while rests of tags are filtered out. Even static tags contain dynamic content, which are filtered out by JavaScript Tester. After filtering HTML, content is converted into DOM. It includes parse tree generator at client side browser to reduce the anomalous behavior of browsers. It is restricted to server side only and browser source need to be modified for obtaining results.

Choudhary & Dhore [13] proposed code injection detection tool based on a Proxy Agent, which classifies the request as scripted request, or query based request. There are two modules: Query Detector and Script Detector. The HTTP request coming from client side is first send to the CIDT within which the request is passed to both modules one by one. Firstly, the Query detector validates the request and the query is rejected if any invalid character is found. Only

the valid requests are passed to the next module, the Script detector that also filters the request for invalid tags and encodes it before forwarding it to the web server. The disadvantage of this approach is it requires more time to response that is the delay time is more.

Matsuda & Koizumi [14] suggested a detection algorithm against cross-site scripting attacks by extracting an attack feature of XSS by considering the appearance position and frequency of symbols. It learns the attack features from given attack samples. In this three modules are presented. The first one is the classification module in which sample of 32 characters is gathered based upon the characters that occur frequent in attacks. The second module will calculate the important degree of characters. The final module will detect the attack-by-attack feature value & threshold that is taken as 15 for the proper detection of the attack in proposed approach. The main disadvantage is it is calculation based and it does not tackle the new attacks effectively.

Elhakeem & Barry [15] broach on the issues surrounding cross-site scripting attacks and providing a simple and useful security model to protect websites from such attacks using ZEND framework application. The security model is based on a chain of levels and is built using a combination of tools. It is divided into four levels as: Security Awareness, Server Security, Client Security, & Design Guidelines. The framework described by them is Zend Framework (ZF), which is an open source framework for developing web applications and services with PHP. This loosely coupled architecture allows developers to use components individually and offers a robust Model View Controller (MVC) implementation. The MVC paradigm breaks the application's interface, into three parts which are: Model: The model part of the application is the part that is concerned with the specifics of the data to be displayed, View: The view consists of bits of the application that are concerned with the display to the user, & Controller: The controller ties together the specifics of the model and the view to ensure that the correct data is displayed on the page. It accepts input from the user and instructs the model and view port to perform actions based on that input. It requires lot of tools to be combined so compatibility issues are there.

5. PROPOSED APPROACH

Cross-Site Scripting is one of the most dangerous and the common attacks found over the web applications. This survey presents study of the ongoing techniques against XSS attacks. These techniques suffer from the following weaknesses:

- Built-in limitations
- Partial implementations
- Complicated framework
- Developer's ability
- Run-time overhead
- False positives and false negatives
- Insecure channel between the web server and web browser
- Response delay
- Additional infrastructure
- Cost of deployment
- Don't prevent DOM based attacks

Our proposed system will try to remove almost all of these weaknesses. It will include a two-tier approach- one for detecting persistent and non-persistent XSS attacks and second for prevention of DOM based XSS attacks. For the first tier we will implement our logic of script guard in the controller part of MVC2 architecture of server. The controller receives all requests from the clients & forwards those requests to the respective pages as per request. The controller receives parameters sent by the client and scans these parameter values for suspected XSS attacks. These values are matched with sets of expressions where every expression match means an attack. In case of an attack, the requests do not go beyond the controller and the client is redirected back to the page where he requested. For the second tier that is for prevention of DOM based attacks we will have a small JavaScript code (DOM attack detector script) which is sent to the client with every response. This code acts only at client side and will prevent any sort of DOM based XSS attacks. Thus, our proposed work will detect all types of XSS attacks. Even it will have a little performance overhead but it will have a minimum response delay. There is no need of additional infrastructure and have not a complex framework.

6. CONCLUSION

XSS attack is one of the most common and dangerous web application attacks that can reveal information about a user or company profile. This paper presented what XSS attacks are, what are there types, the previous approaches for prevention of these attacks with there limitations. Then we showed our proposed approach and how it is better.

Many industries are employing web services for their benefits on the World Wide Web but for relieving themselves from the additional cost, they do not go for the security of the websites they created. Eventually it harms the users and company too. With the expansion of web applications, it is urgency to have an comprehensive and coherent structure for the prevention of unified XSS and other important web application attacks.

7. REFRENCES

[1] http://www.ponemon.org/blog/live-threat-intelligence-impact-report-2013-1

[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[3] http://info.cenzic.com/rs/cenzic/images/Cenzic-Application-Vulnerability-Trends-Report-2013.pdf

[4] Jeom-Goo Kim," Injection Attack Detection using the Removal of SQL Query Attribute Values", IEEE, pp. 1-7, 2011.

[5] Martin Johns, Bjorne Englemann, Joachimm Posegga,"XSSDS: Server-side Detection of Cross-site Scripting Attacks", Annual Computer Security Applications Conference, IEEE, pp. 335-344, 2008.

[6] http://en.wikipedia.org/wiki/Cross-site_scripting

[7] Gary Wassermann, Zhendong Su, "Static detection of cross-site scripting vulnerabilities", ACM/IEEE 30th International Conference on Software Engineering, ICSE '08. pp. 171-180, 2008.

[8] Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda, and Christopher Kruegel," SWAP: Mitigating XSS Attacks using a Reverse Proxy", ICSE Workshop on Software Engineering for Secure Systems, IEEE, pp. 33-39, 2009.

[9] Qinglin Wu, Yanzhong Hu, Yan Wang, "Unit Testing and Action-Level Security Solution of Struts Web Applications Based on MVC", International Conference on Biomedical Engineering and Computer Science, IEEE, pp. 1-4, 2010.

[10] E. Galan, A. Alcaide, A. Orfila, J. Blasco," A Multi-agent Scanner to Detect Stored-XSS Vulnerabilities", International Conference for Internet Technology and Secured Transactions (ICITST), IEEE, pp. 1-6, 2010.

[11] Rattipong Putthacharoen, Pratheep Bunyatnoparat, "Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique", 13th International Conference on Advanced Communication Technology (ICACT), IEEE, pp. 1090-1094, Feb 2011.

[12] Sharath Chandra V., S. Selvekumar, "BIXAN: Browser Independent XSS Sanitizer for Prevention of XSS Attacks", ACM SIGSOFT, Volume 36 Number 5, September 2011.

[13] Atul S. Choudhary And M.L Dhore, "CIDT: Detection Of Malicious Code Injection Attacks On Web Application", International Journal Of Computing Applications Volume-52-N0.2, pp. 19-25., August 2012.

[14] Takeshi Matsuda, Daiki Koizumi, "Cross Site Scripting Attacks Detection Algorithm Based on the Appearance Position of Characters", 5th International Conference on Communications, Computers and Applications, IEEE, pp. 65-70, October 2012.

[15] Yousra Faisal Gad Mahgoup Elhakeem , Bazara I. A. Barry," Developing a Security Model to Protect Websites from Cross-site Scripting Attacks Using Zend Framework Application", International Conference on Computing, Electrical and Electronics Engineering (ICCEEE), pp. 624-629, August 2013.