STM JOURNALS

Journal of Artificial Intelligence Research & Advances
ISSN: 2395-6720 (Online)
Volume 6, Issue 1
www.stmjournals.com

Web Abuse Using Cross Site Scripting (XSS) Attacks

Mohd Umar John*, Junaid Latief Shah, Gazi Imtiyaz Ahmad
Department of Information Technology, Sri Pratap College, Srinagar, Jammu and Kashmir, India

Abstract

In today's modern world, most of the applications are using World Wide Web (www) for information processing and transaction management. The popularity of web has eased out global outreach and accessibility to different users around the globe. Although web traffic has scaled up, it has also increased the abuse of applications by malicious html-based attacks by users; one such attack being the cross-site scripting (XSS). This attack poses a serious threat to web applications and e-databases that may include sensitive user data. Although other web attacks like SQL Injection, CSRF, phishing and session hijacking are also common, XSS tops the list of preferred technique for hackers to capitalize web resources for malicious activities. In this paper, we draw an overview of XSS attacks and its different types. We also discuss certain code prevention techniques possible including robust defense mechanisms. The paper also explicates discussion over the related work that has been concluded by researchers for mitigation scenario and techniques possible for prevention.

Keywords: *XSS, SQL, WWW, CSRF*

****Author for Correspondence*** E-mail: mohdumarjohn@gmail.com

INTRODUCTION

The web is one of the greatest innovations of human race and has originated as a substantial interface in our day to day activities. The applied domain of information and communication technology that deals with the security of web sites, web application and web services is referred to as web security. In older days, corporate software code would reside in secure areas of the company's intranet. An office application would install on a single system or all independent machines along with their software copy. For sharing resources, these systems would communicate with each other using the services of a LAN. As such, there were minimal chances of any malicious activities carried out by the hackers. Given the today's changing world scenario, every software application has migrated to the web [1]. The web services and application popularity is increasing exponentially given the origin of high speed broadband networks including mobile internet such as 3G/4G. Today, the web applications have become backbone for communication between the client and the service providers. This web-based communication is preferred, given the popularity of beautiful and user interactive web pages which are developed by extensive usage of some client side scripting languages such as javascript, vbscript, jquery [2]. As such, their extensive usage poses some serious threats and vulnerabilities to the web applications such as code injections like CSRF, SQL injections and XSS, with later being the top most threat according to OWASP [3]. The database injection, also known as SQL injection, is a technique wherein victim of the attack is a database residing on a server machine. The attacker injects code into the database formed as a part of an SQL query and masquerades in to sneak user data such as passwords, usernames etc. According to research conducted by web application security consortium, SQL injection forms a total of 35% of hacking incidents reported around the world [4]. Recent independent research conducted by Acunetix labs report that more than 50% of web applications scanned every year are susceptible to SQL injections [5]. The SQL injection attack comes in various variants such as blind SQL injection whose task is to query database true or false questions, and based on the response of web applications determines the answer. This attack is synonymous with hit

and trial method wherein hacker tries to sneak in access to the victim database. The SANS institute reports that 60% of internet attacks are directed towards the web applications [6]. Survey conducted by computer security institute (CSI) concludes that defacing of web sites is a common problem for many organizations, since 10% of such incidents were reported by 92% of the respondents. Another independent research by Gartner team shows that 75% of cyber attacks are directed towards web-based applications. Cross site scripting (XSS) is an attack that varies from SQL injection in the way that XSS prime target is the client browser, whereas in SQL injection, the victim is the database server. The major cause of cross site scripting attacks is the incorrect coding techniques applied by developers and extensive use of scripting languages for interactive web pages. Non-validation of user input and unrestricted user input has also contributed towards XSS and its variant attacks.

CROSS-SITE SCRIPTING ATTACK (XSS)

The client-side code injection attack refers to cross site scripting (XSS) attack wherein attacker induces malicious scripts into the web application. This is also known as malicious payload [7]. XSS is considered to be most common and dangerous attack among the web application vulnerabilities. The common cause of XSS is the use of un-validated or un-encoded user input into the web applications. The attack abuses vulnerable code of your web applications permitting a hacker to induce malicious script into a web form to allow them to gain authorization or deface your application i.e. improper sanitization or filtering. The malicious XSS executable code is normally written in standard scripting and programming languages like javascript, vbscript and php. In XSS, the attacker exploits vulnerability within a web application instead of direct victim targeting [8]. As per recent surveys, XSS has become the most common and dreadful security problem that any website or web application could face. As per analysis of western association of schools and colleges (WASC), reports indicate that more than 100,059 XSS threats have identified which have attacked more than 31,373 websites. The web applications wherein XSS attacks have been reported commonly including Facebook, twitter, MacAfee, MySpace, eBay and Google [9].

Types of XSS Attacks

There is no defined standard taxonomy of cross site scripting but generally, researchers classify these attacks in three main attack categories:

1. Type 0 or DOM-based attack;
2. Type 1 or Persistent attack; and
3. Type 2 or Non-persistent attack.

Type 0 or DOM Based Attack

This type of cross site scripting attack is also known as document object model based attack. In this attack, the security loophole exists in client-side script within a web page. As an example, consider if javascript code executes an object model like *document.url* and uses this to post back some HTML code to its own web page. If HTML tags defining this data are not efficiently sanitized, a likelihood of XSS security vulnerabilities may always occur. This written script will be re-executed by client browsers as HTML which could involve extra client-side script.

Type 1 or Persistent XSS Attack

These are also called stored XSS or type 1 attacks. This attack is persistent, because it gets stored somewhere on the server and its effect on the victim machine is not immediate. This attack stores malicious script on the server machine (usually in database) and is reflected back as normal script to all users. As an example, consider a public forum wherein users write an HTML formatted post or comments on a post for other online users to read. When some victim reads the post, the code gets executed inside the victim's browser and does undesirable operations such as session hijacking, url redirection etc. The operation is shown in the Figure 1 below.

For example; the malicious code could be posted like this:
<b> Welcome to portal <script> window.location.href="http://hackerlocation.com"</script> .The above comment persists in

database and at a later stage, when victim visits the page; the same is displayed to victim. The code inside script tag gets executed and the victim will be redirected to "hackerlocation.com".

Type 2 or Non-Persistent Attacks

Non-persistent or Type 2 attacks are also referred to as reflected attacks and occur when data supplied by user via form submission occurs or during query, string value is instantly processed by the server to transmit back data to the given user. In other words, the user data is reflected back unaltered. If the victim data is not properly sanitized, this could result in execution of client side inside user's browser.

XSS DEFENSE APPROACHES

The analysis of popular web application attacks reveals that root cause of XSS attacks is the improper handling of data inputs and incapability to screen or filter out data request originating into web. Therefore the major factor is inability to sanitize user input [10, 11]. The solution to this issue is the mitigation strategy applied at the server end or client end [8]. The mitigation scenario at the client side involves validation of data input. The goal is to limit the end user from adding data to the fetched web page. Contrary to this, server side solutions include client side techniques as well as output sanitization. One not so popular strategy could be to restrict or obstruct javascript execution in the browser [12]. However, this approach is pessimistic in the sense that it will restrict a user from viewing or building up interactive web pages. Some other solutions suggest using inbuilt browser plug in to secure data. Using efficient coding practices sanitize and validate improper data input which provides robust way to eliminate data vulnerabilities. There are number of ways by which input sanitization could be handled. The most common methods include replacement and elimination techniques for search of blacklisted characters [1]. Replacement techniques simply replace code with non-malicious characters while as elimination simply removes them. Other techniques include restricting data input to white listed characters only. Some other characters such as escape sequences that generate specific patterns and meanings in client side engines need to be searched and accordingly removed. Efficient coding is the substantial technique for removing XSS vulnerabilities apart from being computationally intensive and error inductive by human beings. Some other defense mechanisms as recommended by OWASP include:

Specifying a Char-Set: The users should ensure that all web application pages specify a UTF-8 char-set in the headers and in the beginning of the HTML head element.

Escaping HTML: All the user input must be HTML escaped. This means that characters such as <,>, &," should be replaced with &lt, &gt, &amp and &quot respectively. To replace single quoted HTML properties; the guideline is to replace single quote (') with &#39. A number of scripting languages such as jsp, asp and php support this feature by providing well defined functions. Another aspect of XSS defense is the detection of XSS vulnerabilities that lays emphasis on identifying server side script vulnerabilities. Approaches such as static analysis show negative vulnerabilities but on the other side generate many false positives. Recent advances in detection techniques combine the use of both static analysis as well as dynamic analysis methods for accuracy improvisation. Static code analysis approach is an XSS vulnerability detection method used in web applications. Its basic operation involves tracing data transmission between the source and legitimate destination. If malicious data succeeds in reaching the destination system, it could result in XSS attack. A number of approaches related to static code analysis have been proposed; some of which have been implemented while others are still to be tested in real web environment.

EXISTING RESEARCH WORK

A majority of cross site scripting attacks has been discussed by researchers over the period of time, but arriving at a substantial solution has yet not been achieved. As already discussed above, vulnerabilities are mitigated either at the client side or server side. Authors

in [12] suggested to detect vulnerable javascript in Firefox browser, proposed a client side solution which is based on an auditing system for misuse detection. Their proposed system checks and compares malicious javascript against some abnormal activity which is based on certain set of rules. For detection of input validation errors, some researchers recommend using static analysis strategy [10, 11, 7, 13]. The only drawback to this approach is the access to the application source code. Also, these techniques must usually have support of dynamic analysis technique to make the system robust. In [14] authors used this idea to determine runtime behavior of the application to detect XSS, number of systems have been developed and are available. These include Intrusion detection systems (IDS) [8, 15], reversal proxies [16] and application level firewalls [17]. The intrusion detection systems allow recognition of certain traffic patterns against set of predefined rules that detect XSS attacks. Reverse proxies search for unauthorized and injected code in received responses from any web application. The firewalls act as a gateway and control data transmission across malicious domains in addition to critical information tracking. Authors in [17] proposed a client based web proxy known as Noxes which serves as a firewall at application level and also regulates web traffic. The only issue with it is user involvement or action rules, whenever a malicious activity is recorded. Authors in [18] suggested, observing information leakage based on client side approach of input data tainting in web browser. Their proposed approach monitors sensitive information flow inside the browser and thus succeeds in blocking XSS attacks at the client side. The end user is alarmed and provided an option before any transfer of sensitive information. If end user approves request, only then information is transmitted. This feature supplements protection level of user while browsing any web application. The entire client based solutions mentioned above and in the literature suffer from one common drawback, i.e., application updates need to be installed individually on every node. Having this drawback is the only reason of holding back and not recommending client side solutions. Table 1 summarizes XSS defense mechanisms extracted from available literature.

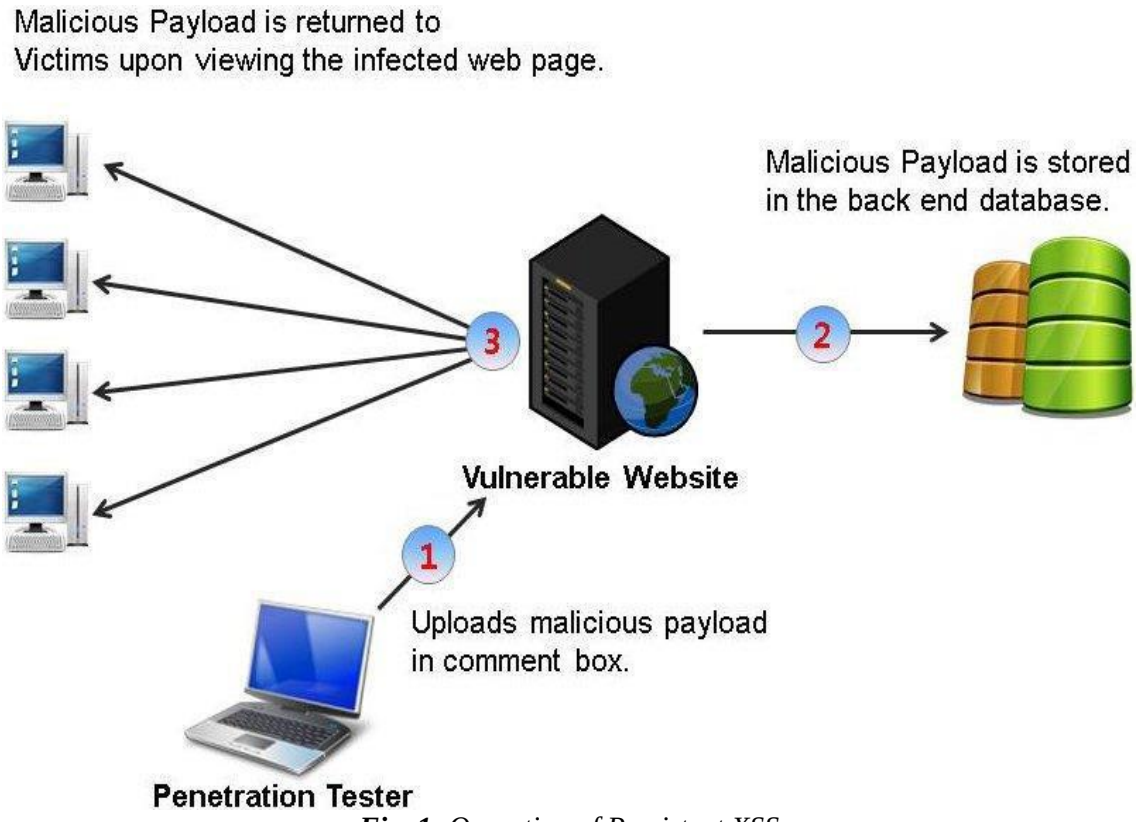

Fig. 1: *Operation of Persistent XSS*

Table 1: *Review and Summary of XSS Defense Mechanisms.*

Paper Reference	Domain	Merits	Drawbacks
Kirda *et al*. [17]	Web browser	Supports mitigation practice of XSS attack that remarkably weakens the connection alert prompts and providing the defense against XSS attacks	The low reliability and prohibition of simple HTML usage.
Johns [19]	Web server	This tool does not demand any changes in the source code of web applications and defense towards XSS attacks from server side.	They require highly structured Server domains method that are unfit to handle. Restricted security is offered for accessing sensitive resources of a web application like cookies
Kals *et al*. [20]	Web server	Focuses on the application level threats in the real-time websites human intervention	More than 25000 live web pages were scanned and tested by the authors and no substantial proof is available for these web applications.
Johns *et al*. [15]	Web server	XSS attacks are measured by the deviation of HTTP web request and its related HTTP response.	Advanced training period for acquiring more data scripts suffers from few false positives for discovering persistent XSS attack
Bisht *et al*. [21]	Web server	Technique is very powerful for the detection of unlicensed script data involved in HTTP response web page.	It has drastic influence on the web browser parsers to deduce vulnerable HTML data and are susceptible to threats involving browser parse quirks.
Wurzinger *et al*. [16]	Web proxy /Web server	It detects the divergence between simple and injected Javascript code.	This technique cannot detect all kinds of XSS attacks.
Ter *et al*. [22]	Web server /Web browser	It delivers defense against suspicious script injections, and enables aid for script-less HTML content. The secure development of the intended HTML parse tree on the web browser is also guaranteed by it.	Modification is required both at the client as well as server side and it does not provide any idea for handling the unsafe HTML content at the server side.
Van and Chen [23]	Web server	This technique enables web clients to classify among trusted and untrusted data to prevent exploitation of XSS vulnerabilities	User input sanitization is very difficult.
Shahriar *et al*. [24]	Web server	The concepts ‘‘boundary injection’’ and ‘‘policy generation’’ are its main strengths.	It requires large amount of time for the detection of the attacks.
Shahriar *et al*. [25]	Web server	Random token generation and characteristics of simple Javascript code are its chief strengths.	It is able to discover part of injection attack code only.

CONCLUSION

Given the data traffic and migration of application to web, cross site scripting attacks are a nightmare and postulate a critical threat to sensitive data and user privacy. Although other web application attacks are also significant, XSS still finds number one place in the list of critical vulnerabilities as provided by OWASP.

In this paper, we introduced XSS attack problem and discussed its different categories. We also provided an overview of certain substantial techniques used for robust defense mechanisms. The paper also provides critical review of related work that has been carried out by researchers for mitigation scenario and techniques possible for prevention. To completely secure a web application is not possible given the number of advanced malicious activities carried over internet. However, given today’s advanced technology including machine learning algorithms, some novel approaches and techniques have come up for securing web which need to be analyzed and discussed.

REFERENCES

1. Shar LK, Tan HBK. Defending against Cross-Site Scripting Attacks. *Computer*. 2012; 45(3): 55–62p.
2. Johari R, Sharma P. A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, IEEE. May 2012; 453–458p.

3. OWASP T. Top 10–2013. *The Ten Most Critical Web Application Security Risks*. 2013.
4. Web Application Security Consortium. WASC Threat Classification. *Release, Web Application Security Consortium*. 2010.
5. Calin B. Email Header Injection Web Vulnerability–Acunetix. 2013. *URL https://www. acunetix. com/blog/articles /email-header-injection-web-vulnerability-detection*.
6. Allen M. *Social Engineering: A Means to Violate a Computer System*. SANS Institute, InfoSec Reading Room; 2006.
7. Jovanovic N, Kruegel C, Kirda E. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In *Security and Privacy, 2006 IEEE Symposium on*. May 2006; 6p.
8. Kruegel C, Vigna G. Anomaly Detection of Web-Based Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*. Oct 2003; 251–261p.
9. Xie Y, Aiken A. Static Detection of Security Vulnerabilities in Scripting Languages. In *USENIX Security Symposium*. Jul 2006; 15: 179–192p.
10. Wassermann G, Su Z. Static Detection of Cross-Site Scripting Vulnerabilities. *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, New York, USA. 2008; 171–780p.
11. Wassermann G, Su Z. Static Detection of Cross-Site Scripting Vulnerabilities. In *Proceedings of the 30th International Conference on Software Engineering*, ACM. May 2008; 171–180p.
12. Hallaraker O, Vigna G. Detecting Malicious Javascript Code in Mozilla. In *Engineering of Complex Computer Systems, 2005, (ICECCS 2005) Proceedings, 10th IEEE International Conference on*, IEEE. Jun 2005; 85–94p.
13. Huang YW, Yu F, Hang C, *et al*. Securing Web Application Code by Static Analysis and Runtime Protection. In *Proceedings of the 13th International Conference on World Wide Web*, ACM. May 2004; 40–52p.
14. Balzarotti D, Cova M, Felmetsger V, *et al*. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. May 2008; 387–401p.
15. Johns M, Engelmann B, Posegga J. Xssds: Server-Side Detection of Cross-Site Scripting Attacks. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, IEEE. Dec 2008; 335–344p.
16. Wurzinger P, Platzer C, Ludl C, *et al*. SWAP: Mitigating XSS Attacks Using a Reverse Proxy. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*. IEEE Computer Society; May 2009; 33–39p.
17. Kirda E, Kruegel C, Vigna G, *et al*. Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks. In *Proceedings of the 2006 ACM Symposium on Applied Computing*. Apr 2006; 330–337p.
18. Vogt P, Nentwich F, Jovanovic N, *et al*. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *NDSS*. Feb 2007; 2007: 12p.
19. Johns M. SessionSafe: Implementing XSS Immune Session Handling. In *European Symposium on Research in Computer Security*, Springer, Berlin, Heidelberg. Sep 2006; 444–460p.
20. Kals S, Kirda E, Kruegel C, *et al*. Secubat: A Web Vulnerability Scanner. In *Proceedings of the 15th International Conference on World Wide Web*. May 2006; 247–256p.
21. Bisht P, Venkatakrishnan VN. XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Berlin, Heidelberg. Jul 2008; 23–43p.
22. Ter Louw M, Venkatakrishnan VN. Blueprint: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers. In *Security and Privacy, 2009 30th IEEE Symposium on*. May 2009; 331–346p.
23. Van Gundy M, Chen H. Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks. In *NDSS*. Feb 2009.

24. Shahriar H, Zulkernine M. S2XS2: A Server Side Approach to Automatically Detect XSS Attacks. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE 9th International Conference on*. Dec 2011a; 7–14p.
25. Shahriar H, Zulkernine M. Injecting Comments to Detect JavaScript Code Injection Attacks. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*. Jul 2011b; 104–109p.

Cite this Article
Mohd Umar John, Junaid Latief Shah, Gazi Imtiyaz Ahmad. Web Abuse Using Cross Site Scripting (XSS) Attacks. *Journal of Artificial Intelligence Research & Advances*. 2019; 6(1): 69–75p.