4th International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB-18)

Reducing attack surface corresponding to Type 1 cross-site scripting attacks using secure development life cycle practices

Syed Nisar Bukhari
NIELIT Srinagar
J&K, India
nisar@nielit.gov.in

Muneer Ahmad Dar
NIELIT Srinagar
J&K, India
muneer@nielit.gov.in

Ummer Iqbal
NIELIT Srinagar
J&K, India
ummer@nielit.gov.in

***Abstract*—While because the range of web users have increased exponentially, thus has the quantity of attacks that decide to use it for malicious functions. The vulnerability that has become usually exploited is thought as cross-site scripting (XSS). Cross-site Scripting (XSS) refers to client-side code injection attack whereby a malicious user will execute malicious scripts (also usually stated as a malicious payload) into a legitimate web site or web based application. XSS is amongst the foremost rampant of web based application vulnerabilities and happens once an internet based application makes use of un-validated or un-encoded user input at intervals the output it generates. In such instances, the victim is unaware that their data is being transferred from a website that he/she trusts to a different site controlled by the malicious user. In this paper we shall focus on type 1 or "non-persistent cross-site scripting". With non-persistent cross-site scripting, malicious code or script is embedded in a Web request, and then partially or entirely echoed (or "reflected") by the Web server without encoding or validation in the Web response. The malicious code or script is then executed in the client's Web browser which could lead to several negative outcomes, such as the theft of session data and accessing sensitive data within cookies. In order for this type of cross-site scripting to be successful, a malicious user must coerce a user into clicking a link that triggers the non-persistent cross-site scripting attack. This is usually done through an email that encourages the user to click on a provided malicious link, or to visit a web site that is fraught with malicious links. In this paper it will be discussed and elaborated as to how attack surfaces related to type 1 or "non-persistent cross-site scripting" attack shall be reduced using secure development life cycle practices and techniques**

Keywords—cross-site scripting; XSS; non-persistent; attack

I. Introduction

To develop rich Web applications, the most prominent tool that we use is the JavaScript. The dynamic nature of Web applications would not be possible without the execution of client-side code embedded in HTML and XHTML pages. But unfortunately, whenever you add complexity to a system, you also increase the potential for security breaches and embedding JavaScript to a Web application is no exception. The most common problems introduced by JavaScript are:

1. A malicious web based application using JavaScript may try to do changes to the local system, such as copying or deleting files.
2. A malicious web based application using JavaScript may try to monitor activity on the local system, such as keystroke logging.
3. A malicious web based application using JavaScript may try to interact with other web applications the user has open in other windows or tabs.

The first two problems can be mitigated by turning the browser into a type of "sandbox" which limits the way JavaScript is allowed to work so that it only works within the browser's world.

However the third can also be limited somewhat as well, but it is all too easy to get around that limitation because whether a particular page can interact with another page in a given way might not be something which can be controlled by the software employed by the very end user. Sometimes, the ability of one web application JavaScript to steal data meant for another web application can only be limited by the due care of the other website's developers.

Checking for various vulnerabilities and providing solution for it is an important part for any web application[8], [9], [10]. The threats under consideration are XSS (Cross Site Scripting) Attack [11]. XSS is an attack which concentrates on embedding unwanted data on the valid web application. This inserted data can be a URL link to attacker's web application, using which attacker can trick the user to give away it's crucial data.

SQL injection is a code injection attack in which attacker targets the database of the system to access data from it or to change the valid data with ambiguous data of his/her own [12].So the main idea behind the cross-site scripting is in the fact that vulnerabilities in a given web applications use of dynamic web design elements may give someone the chance to use JavaScript for security compromises.

This is called "cross-site" as it involves interactions between two separate web applications to achieve its goals. However in many instances, even though the exploit involves the use of JavaScript, the web application which is vulnerable to XSS exploits does not have to employ JS itself at all. In the case of local XSS exploits only, does the vulnerability have to exist in JS sent to the browser by a legitimate web application. [1]

978-1-5386-4606-9

II. Type 01 or Non persistent XSS attack scenario

In reflected or type 1 cross site scripting attack type, the malicious code is a part of the victim's request to the web application. The web application then embeds this malicious code in the response object sent back to the user. The below diagram illustrates the scenario:

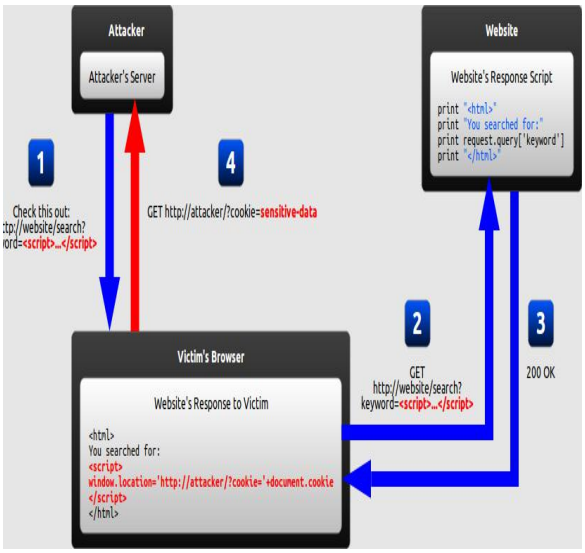

Fig. 1. Non Persistent XSS attack scenario

1. The attacker creates a URL containing a malicious code and sends it to the target victim.
2. The target victim is tricked by the malicious user or attacker into requesting the URL from the web application.
3. The web application includes the malicious code from the URL in the response object

The target victim's browser interprets the malicious code script inside the response, sending the target victim's cookies to the attacker's server.

These two ways are similar, and can be more successful with the use of a URL shortening, which would mask the malicious string from users who may otherwise identify it.[2]

III. Reflected Cross site Scripting attack

At first, the reflected Cross site Scripting attack might seem harmless as it requires the target victim himself to actually send a request containing a malicious code. As nobody would willingly attack himself, there seems to be no way of actually executing the attack. So, there are at least two common ways of causing a target victim to launch a reflected Cross site Scripting attack against himself:

- The attack occurs when an malicious user takes advantage of such applications and creates a request object with malicious data that is later presented to the user requesting it. The malicious data is usually included into a hyperlink, positioned so that the user will come across it in a web application, a message board, an email, or an IM.[5]
- A user targets a large group of people, the malicious user can publish a link to the malicious resource (on his own web application, on any social network) and wait for visitor to click it.

IV. Statistics showing the frequency of Type 1 XSS attack

Web applications continue to fall prey to Cross site Scripting attacks because most needs to be interactive, accepting and returning data from users [3]. As per the report by Search Security organization team the penetration testing work conducted at Intelguardians, approximately 80% of the Web applications they tested have XSS flaws [6]. Most major web applications like Google, Yahoo and Facebook have all been affected by XSS attacks at some point. The latest report by WhiteHat Security Statistics reveal that 86% of all websites had at least one serious vulnerability but XSS is the most frequently found serious kind of vulnerability. Of the total population of the vulnerabilities identified, Cross-Site Scripting, Information Leakage and Content Spoofing took the top three spots at 43%, 11% and 13% respectively. This is near linear repeat of 2011 where the percentage was 50%, 14% and 9% [4]

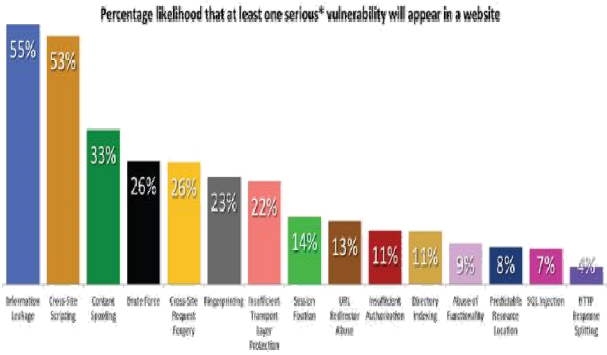

Fig. 2. Overall vulnerability population (2017).%age breakdown of all serious vulnerabilities discovered

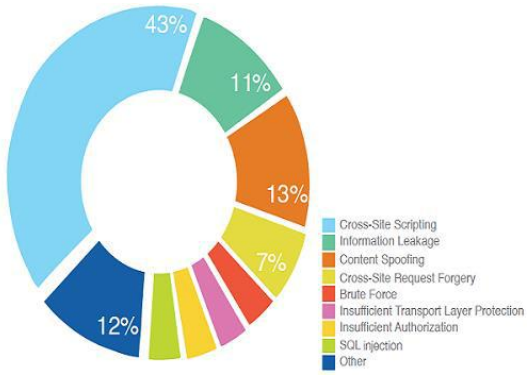

Fig. 3. Top 15 vulnerability classes (2017) -sorted by vulnerability class.

978-1-5386-4606-9©2018 IEEE

V. Research Findings

Most of the web developers particularly those having less experience or no experience at all write in secure code. I reviewed as part of my work the source code of around ten web applications and I found that web developers are usually not aware that their web applications are open to script injection attacks. Whether the purpose of an attack is to deface the web application, or to execute client side script to redirect the user to a malicious users' site, script injection type attacks is a problem that Web application developers must take care of. Clint side script injection attacks is a concern of all web application developers, whether using ASP.NET, ASP, or other web development technologies. The most great thing about the Microsoft ASP.NET web technology is that request validation facility prevents these attacks by not allowing un-encoded HTML/XHTML content to be processed by the server unless the web developer decide to allow that type of content.

I asked the developers for testing and I have found that if they enter something which includes html tags or like tokens as <h1> hello</h1> in an input field such as text box, they get the HttpRequestValidationException because of the ValidateRequest option which is a part of the built-in protection mechanism with ASP.NET. This feature can be enabled on a per-page basis, or globally through web.config file settings. This option, when set to "true," instructs ASP.NET to inspect all inputs into a Web-based application for potentially dangerous inputs. If any potentially dangerous inputs are detected, then HttpRequestValidationException is thrown and the attack is halted. This could be an attempt to compromise the security of your web application, such as a XSS attack. I have experienced that based on the message received after such an attempt they immediately override applications request validations settings by setting the requestValidationMode feature in the httpRuntime configuration to requestValidationMode="2.0".After setting this value in the configuration and then disabling request validation by validateRequest="false" in the Page directive , the request is easily processed which definitely is a security threat. So it is highly recommended that your web application explicitly makes check of all inputs in this case.

Caution: whenever the request validation feature is disabled, information can be submitted to your web application; it is the duty of the web application developer to make sure that content is properly encoded. So to reduce the risk from cross-site scripting attacks, developers need to transform or neutralize user input that may contain potentially executable code or script into non-executable forms. That is, the Web browser needs to be told in some way that the following data is not executable code and should treated as data only. The way this transformation or neutralization is achieved is through encoding. The process of encoding will automatically replace '<' or '>' with their corresponding HTML encoded representations. For example, '<' may be replaced by '<' and '>' by '>'. Out of ten applications I found only two applications using encoding as Server.HtmlEncode API and writing secure code which clearly states that around 80% of web applications do have XSS and other types of vulnerabilities. This closely matches with the report prepared by Search Security organization team and WhiteHat Security.

VI. Reducing the Exposure using secure development practices

There are several measures you can take as a developer to reduce the exposure to cross-site scripting attacks conducted through your Web-based applications.

- The first defensive measure which can be applied to address a majority of application security vulnerabilities is input validation. Ensure that all un-trusted inputs into Web-based applications conform to the expected input formats Check for correctness with format, length, type, and range. Example sources of un-trusted input include, but are not limited to, data from users, data from a database, or data from an un-trusted Web service.
- Encode any Web response data that may contain user input or other un-trusted input
- Web-based applications built using Microsoft ASP.NET can leverage built-in protection via the ValidateRequest option.
- Another defensive technique that can be used to help protect web applications from XSS attacks is Microsoft Anti-Cross Site Scripting Library (AntiXSS). This library provides additional encoding capabilities not provided by the standard encoding libraries included in the .NET Framework [7].
- The .NET Framework has built-in encoding libraries under the class System.Web.HttpUtility. The encoding methods in this class work by looking for specific characters that are common in cross-site scripting attacks and encode them into non-executable forms [7].
- The Microsoft Anti-XSS Library takes a different approach by first defining a set of valid characters, and then encoding characters not in the valid set. Both are effective in reducing exposure to a majority of XSS attacks; however, they differ in the way in which they reduce exposure.

VII. Conclusion

Cross-site scripting vulnerabilities are the most frequently encountered Web-based vulnerabilities today, and have been found on several major websites. These vulnerabilities manifest in Web-based applications whenever best practices, such as input validation, and Web output encoding are not

978-1-5386-4606-9©2018 IEEE

implemented in code. To reduce exposure to these attacks, developers should implement a multi-layer defense strategy that includes coding best practices such as input validation, Web output encoding, and leveraging built-in platform protection. Microsoft has better enabled developers to do so through the guidance, process and tools of the Microsoft SDL.

References

[1] Web refence; www.techrepublic.com

[2] Web refence; http ://excess-xss.com/

[3] Web refence; http ://www.computerweekly.com

[4] Web refence ;White Hat Security, "Website statistics report"

[5] "Cross-site scripting attack prevention", www.imperva.com

[6] "What is new tactics can prevent XSS attacks", www.searchsecurity.techtarget.com

[7] "Microsoft SDL-Developer starter Kit," (Cross site scripting level-200)", www.microsoft.com/SDL

[8] Thankachan, A.; Ramakrishnan, R.; Kalaiarasi, M., "A survey and vital analysis of various state of the art solutions for web application security," Information Communication and Embedded Systems (ICICES), 2014 International Conference on , vol., no., pp.1,9, 27-28 Feb. 2014.

[9] Dukes, L.; Xiaohong Yuan; Akowuah, F., "A case study on web application security testing with tools and manual testing," Southeastcon, 2013 Proceedings of IEEE , vol., no., pp.1,6, 4-7 April 2013.

[10] Teodoro, N.; Serrao, C., "Web application security: Improving critical web-based applications quality through in-depth security analysis," Information Society (i-Society), 2011 International Conference on , vol., no., pp.457,462, 27-29 June 2011.

[11] OWASP, "https://www.owasp.org/index.php/Top 10 2013-Top10".

[12] A. Tajpour, M. Massrum, "Comparison of SQL Injection Detection and Prevention Techniques", 2nd International Conference on Education Technology and Computer (ICETC), 2010).

978-1-5386-4606-9©2018 IEEE