Impact Factor (SJIF): 4.542

International Journal of Advance Research in Engineering, Science & Technology

e-ISSN: 2393-9877, p-ISSN: 2394-2444

Volume 4, Issue 3, March-2017

A survey on prevention of Cross-Site Scripting

Sweta D. Parmar[1], Ravi K. Sheth[2]

[1]Cyber Security Dept., Raksha Shakti University
[2]Cyber Security Dept., Raksha Shakti University

Abstract — today securing the web application against attacks is a very big challenge. Cross-site scripting (XSS) is a very common type of technique to attack a web application. Cross-site scripting (XSS) is being used by the attackers to steal web browser's cookies or user credentials etc. by injecting the malicious javascript into victim's web application. In order to prevent this type of injection we need to apply input validation as well as output encoding to enforce that the untrusted values are interpreted by the Web browser purely as data and in no circumstances as mark-up or JavaScript code.

Keywords- Cross-site scripting (XSS), prevention, validation, encoding, web application

I. INTRODUCTION

Cross-site scripting (XSS) is a vulnerability in web applications that is caused by insecure coding practices, which do not sanitize data properly. If a user of a vulnerable web application can pass scripts to the server and the server does not sanitize input, then the script can be executed in the user's browser. XSS attacks can redirect a user to a malicious site; it can also steal session ID and authentication cookies allowing an attacker to gain access to restricted resources.
Cross-Site Scripting (XSS) attacks occur in following cases:
Data enters from untrusted source.
Data is sent to a user without being validated for malicious script.

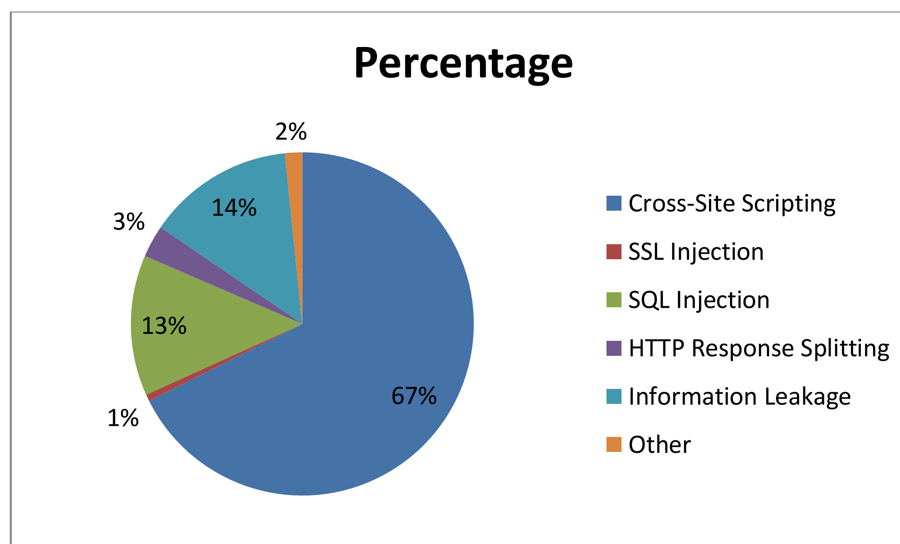

Figure 1 Vulnerabilities Population in 2016

1.1. Threats of cross-site scripting attacks

A. Session hijacking: exploitation of a valid computer session.
B. Misinformation: such as false or incorrect information to a page.
C. Defacing web site: such as adding "You are hacked!" to a page.
D. Phishing attacks: a malicious link in a seemingly legitimate email.
E. Takeover of the browser: Injecting malicious JavaScript code to redirect the user.
F. Flooding: browsers crash or become inoperable.

All Rights Reserved, @IJAREST-2017

II. TYPES OF XSS

Cross-site Scripting can be classified into three categories — Stored XSS, Reflected XSS and DOM-based XSS.

A. Stored XSS

This is the most damaging type of XSS. In Stored XSS attacks an attacker injects a malicious script or payload that is permanently stored (persisted) on the target application (for instance within a database). The popular example of stored XSS is an injection of malicious code in a comment field on a blog or in a forum post.
When a victim goes to the affected web page in a browser, the XSS payload will be taken as a legitimate part of the web page. This causes the victims inadvertently end-up executing the malicious script once the page is viewed in a browser.

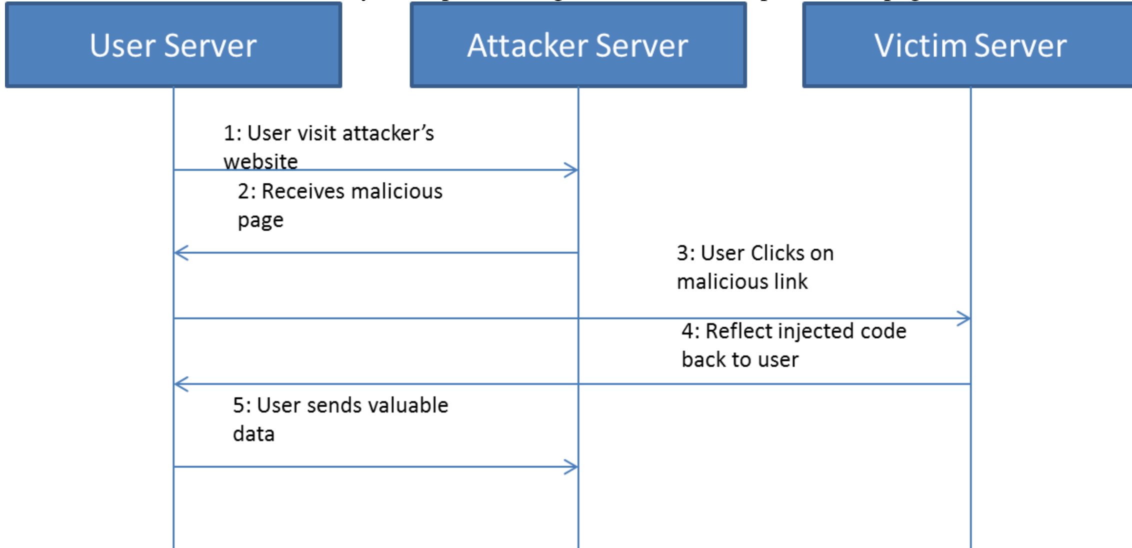

Figure 2 Stored XSS

B. Reflected XSS

The second type of XSS is Reflected XSS. In Reflected XSS, the attacker's payload script has to be part of the request which is sent to the web server and reflected back in such a way that the HTTP response includes the payload from the HTTP request. The attacker uses techniques like phishing and social engineering to tempt the users to accidently make a request to the server containing the XSS payload and execute the malicious script inside the browser. As Reflected XSS is not persisted in the server, the attacker needs to send the payload to each victim. Attackers find social networks very convenient for the spreading the Reflected XSS attacks.

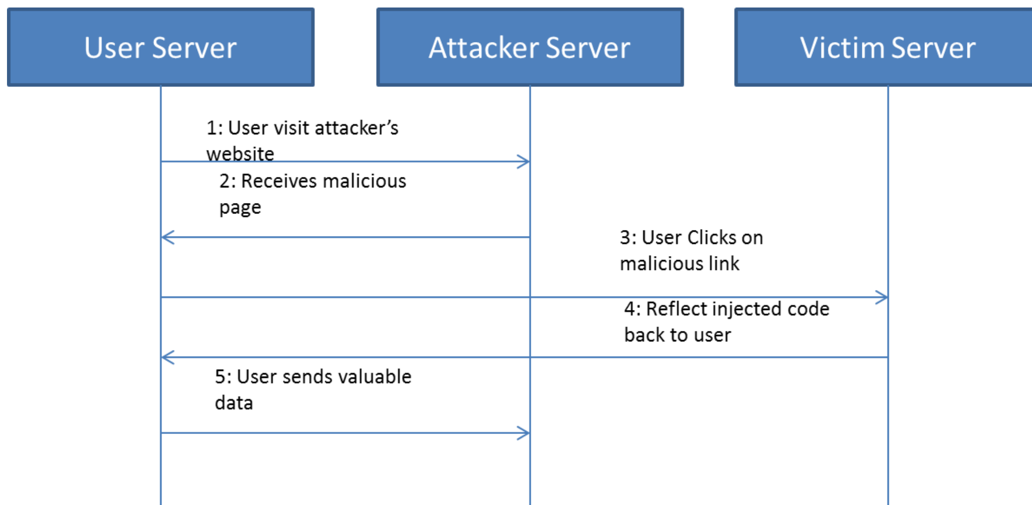

Figure 3 Reflected XSS

C. DOM-based XSS

DOM-based XSS is an advanced XSS attack which is performed when the user provided data of client side scripts is written to the Document Object Model (DOM). Then the web application reads the data from the DOM and sends it to the browser. An attacker can inject a payload if the data is not properly sanitized. The payload will be stored as part of the DOM and executed when the data is read back from the DOM. The DOM-based XSS is generally a client-side attack, and

All Rights Reserved, @IJAREST-2017

the payload is never sent to the server. For Web Application Firewalls (WAFs) and security experts it is very difficult to detect and analyze the server's logs of the DOM-based attack.

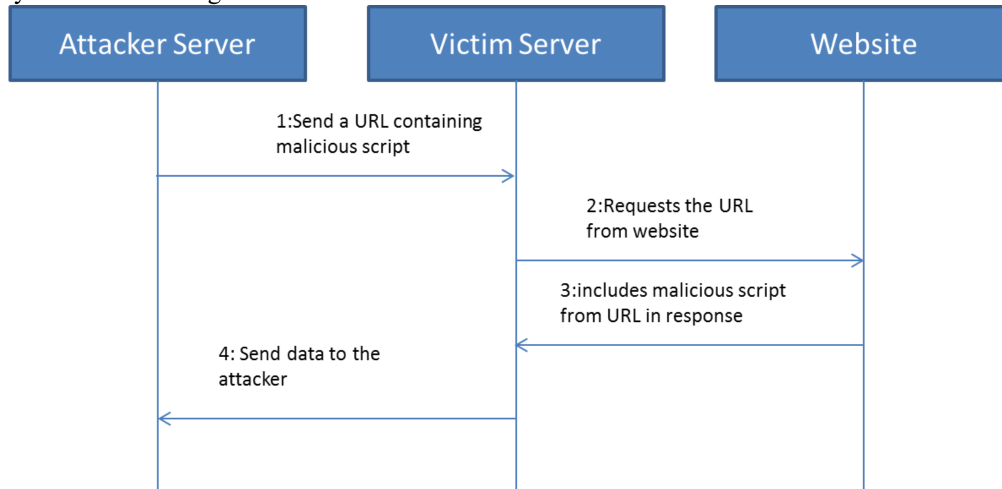

Figure 4 DOM based XSS

III. CROSS-SITE SCRIPTING ATTACK PREVENTION

An XSS attack is a type of code injection where user input is mistakenly interpreted as malicious program or code. In order to prevent this type of attack, secure input handling is needed.

3.1. Prevention methods

For programmers, there are two basic different methods of performing secure input handling.

A. Encoding

Encoding escapes the user input so the browser interprets it as data and not as code. HTML escaping is the most popular type of encoding in web development. It converts characters like < and > into < and >, respectively. If web developer escapes all characters with special meaning, the browser will not take any part of the user input as HTML. While performing encoding in client-side code, the JavaScript language is used, which has built-in functions that encode data for different contexts. It will be possible to inject malicious scripts into some contexts even with encoding. When the user needs to define part of a page's code, at that time encoding is also an insufficient solution. For example the user can define custom HTML in a user profile page. If encoding is performed on this custom HTML, the profile page can consist only have plain text.

B. Validation

Validation is used to filter the user input so that the browser interprets it as code without malicious scripts. Validation is the process in which user input is filtered and all malicious parts of it are removed. It is not required to remove all the code from it. One of the most popular types of validation is allowing some HTML elements such as <em> and <strong> but not allowing others such as <script>. Without concise thought, it looks logical to perform validation by defining a not allowed pattern that should not come in user's input. If a string equivalent to this pattern is found, then it is marked as invalid. Or instead of defining a not allowed pattern, a whitelist approach can be used. It defines an allowed pattern and if a string equivalent to this pattern is not found, then it is marked as invalid. When input has been marked as invalid either it is rejected or sanitised. The input is simply rejected in rejection while invalid parts of the input are rejected in sanitisation. As sanitisation allows a large range of user inputs, it is more useful. While implementing sanitisation, programmer must keep in mind that the sanitisation code itself should not use a blacklisting approach.

3.2. Common features of encoding and validation

These are basically different ways of preventing XSS but they have some common features that are important to understand before using any of them:

A. Context:

Secure input handling needs to be performed differently depending on where the user input is inserted in the page. There are number of contexts in a web page where user input can be inserted. Clearly defined rules must be followed for each of the contexts so that the input inserted by the user cannot go out of its context and be interpreted as malicious script. An attacker can inject malicious code by just inserting the closing delimiter for the context and following it with the malicious code if user input were inserted before first being encoded or validated.

B. Inbound/outbound:

All Rights Reserved, @IJAREST-2017

Inbound means when your web application receives the input and Outbound means before your website inserts the input into a web page. Without conscious thought, it may seem that XSS can be prevented by encoding and validating all inputs as soon as the web application receives them. Any malicious scripts should already have been ineffective whenever they are inserted in a web page. The scripts generating HTML will not have to worry themselves with secure input handling. As described earlier, the problem lies in several contexts in a page where user input can be injected. It is not easy to decide that when user input arrives and which context it will eventually be inserted into, and the same user input needs to be inserted into different contexts frequently. Depending on inbound input handling for prevention of XSS is thus a very delicate solution that will be prone to errors. Inbound validation can be used if we want to add a second layer of protection.

C. Client/server:

The client-side input handling and the server-side input handling, both are needed under different circumstances. Secure input handling must be performed in server-side code to protect against conventional XSS. It can be performed by using any language supported by the server. Secure input handling must be performed in client-side code using JavaScript to protect against DOM-based XSS where the server never receives the malicious code.

V. CONCLUSTION

After going through different papers and articles I have learnt that to prevent XSS Encoding should be the first line of defense, because its main purpose is to make data ineffective so that it cannot be interpreted as code. Almost encoding should be combined with validation. When the input is inserted in a page we know which context to encode and validate for. So the encoding and validation should always be outbound. We should use inbound validation as a second line of defense to sanitize or reject data that is clearly invalid. It cannot provide full security by itself but if at any point outbound encoding and validation is inappropriately performed due to errors it is a useful safeguard. We can protect our web applications website from XSS attacks if these two lines of defense are used accurately. But, due to the difficulty of creating and maintaining the entire website, it is very difficult to achieve full protection against XSS attacks.

ACKNOWLEDGMENT

We express our gratitude and appreciation to all who provided their help and expertise to complete this paper successfully. We are also immensely grateful to Raksha Shakti University for continuous support and motivation.

REFERENCES

[1]. https://help.sap.com/saphelp_nw74/helpdata/en/23/0e94a1e1f54330a6a7c2d8b3183ab1/content.htm
[2]. https://www.cybrary.it/0p3n/an-introduction-to-cross-site-scripting/
[3]. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[4]. http://www.acunetix.com/websitesecurity/xss/
[5]. http://www.webappsec.org/projects/statistics/
[6]. https://excess-xss.com/

All Rights Reserved, @IJAREST-2017