International Journal of Computers & Technology
Volume 4 No. 2, March-April, 2013, ISSN 2277-3061

Web Client and Web Server approaches to Prevent XSS Attacks

Jyoti Snehi[1], *Dr. Renu Dhir*

[1]Chitkara University, Punjab, India

[1]jyoti.snehiverma@gmail.com

[2]NIT, Jalandhar, India

[2]dhirr@nitj.ac.in

ABSTRACT

Websites rely completely on complex web applications to deliver content to all users according to set preferences and specific needs. In this manner organizations provide better value to their customers and prospects. Dynamic websites suffer from various vulnerabilities rendering organizations helpless and prone to cross site scripting attacks. Cross Site Scripting attacks are difficult to detect because they are executed as a background process. Cross Site Scripting is the most common web vulnerabilities in existence today which is most exploited issue .In this paper we have presented various approaches used by clients and Server to prevent XSS attacks

General Terms

Web security, XSS attacks, Web Vulnerability.

Indexing terms

XSS (Cross Site Scripting), Stored XSS, Reflected XSS, DOM based XSS.

INTRODUCTION

XSS or Cross-site scripting is a type of computer insecurity vulnerability found in Web applications which enables attackers to inject client-side script into Web pages and affect other users. Cross-site scripting attacks are essentially code injection attacks, where the attacker s injects code into user's site and therefore can attack any user who visits your supposedly-safe site. It is one of the most prevalent application layer web attacks. XSS targets scripts embedded in a page which are executed on the client side .Figure below describes Basic pattern of XSS attack. [1][2]

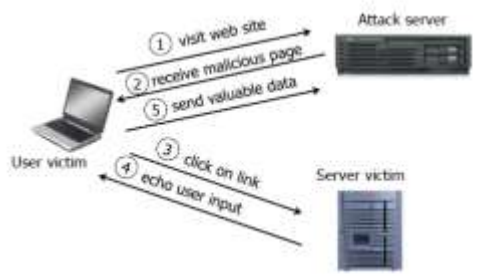

Figure 1: Cross site scripting: Basic Pattern

An XSS attack manipulates content of a Web application and trick users into opening that page. A typical XSS attack work as follows:

1. Form on the page asks user for clicking a link, or entering username or password.
2. User takes the data submitted by the victim and store it in a database
3. User displays that data on the screen to other users.
4. Malicious user submits Script in their form submission, which performs an action when other users visit the page displaying the data they submitted.[3][12]

EVOLUTION OF XSS

XSS, Cross -Site-scripting was reported and exploited in the 1996. Attackers injected malicious code in HTML website. Anyone who visits their website executed this malicious code. In December 1999, David Ross presented how the script is injected into the Server and how code injection work and named it CSS (Cross -Site-scripting). In 2000, Cross-Site-Scripting (CSS) was changed to XSS in order to differentiate from Cascading Style Sheets. In 2004 XSS vulnerability increased to 10.9% and in 2007 it was nearly 21.55 of the total web vulnerabilities. Till Second half of 2007, 80% of all attacks were XSS. According to Web Hacking Incident Database for 2011 (WHID), SQL injection and XSS were the most popular along with other vulnerability that arose as side effect of XSS like Disclosures, Content Spoofing and Stolen Credentials. [4]

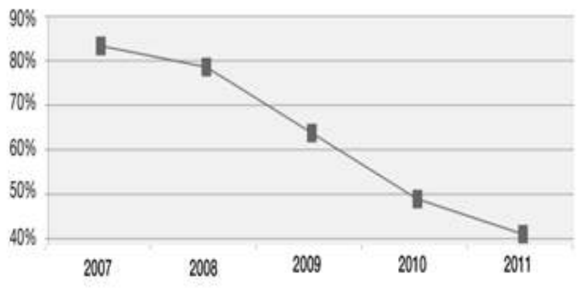

Figure 2: Controlling XSS attacks

ISSUES RELATED TO XSS

XSS Vulnerability

XSS vulnerabilities allow attackers to execute commands and display arbitrary content in a victim user's browser. Cross-site scripting uses vulnerabilities in web-based applications, their servers, or plug-in systems. Attackers attack sensitive information such as cookies and session IDs, reflect the malicious code to victim users, deface or hijack websites, enable malicious phishing attacks, and provide entry points for larger-scale attacks against organization assets and user data. Exploited XSS is used to achieve the Identity theft, Access to sensitive information. Attackers can also manipulate and use other intranet applications. By exploiting XSS vulnerabilities, an attacker can perform malicious actions and Hijack an account. He can spread worms, Access browser history and clipboard contents Control the browser remotely, Scan and exploit intranet appliances and applications. [5][13]

XSS vulnerabilities can be divided into following types:

1. Injection via URL
2. Injection by exploiting client side code
3. Injection by exploiting external feed displayed on a website
4. Injection via permanently displayed data

Cross-site scripting Threats

Cross-site scripting poses severe application risks as the users can unknowingly execute malicious scripts while viewing dynamically generated pages or content provided by an attacker. An attacker can take over the user session before the user's session cookie expires. An attacker can connect users to a malicious server of his choice. An attacker who can convince a user to access a URL supplied by the attacker could cause script or HTML of the attacker's side to be executed in the user's browser. An attacker can take actions with the privileges of the user who accessed the URL by issuing queries on the SQL databases, viewing the results and exploit the known faulty implementations on the target system. [12]Threats related to XSS as a result of its side effect are as follows:

1. Session high jacking
2. Cookie poisoning
3. Malformed URL
4. IFRAME
5. DOS Attack

Cross-Site Scripting Attacks:

Cross-Site Scripting (XSS) attacks occur when data enters a Web application through some untrusted source via a web request. Data is included in dynamic content that is sent to a user without being validated for malicious code. The XSS attacks are broadly classified into three types:

Stored/Non-Reflective/Persistent/ Type-2 Cross Site Scripting

These are those attacks in which the injected code is permanently stored on the target servers in the form of visitor log, database, in a message forum, comment field, etc. An attacker's malicious script is rendered automatically and lure user to a third-party website. These XSS attack embeds the malicious script permanently into the web application. The script wait until people access the page the script is located on. The victim then retrieves the malicious script from the server. It is also known as Non Reflective, Type-2 XSS attack or HTML injection Persistent XSS. [11][12]

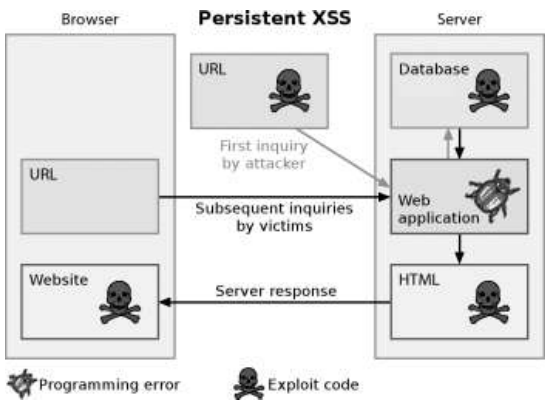

Figure 3: Persistent XSS attack

Non-Stored/Reflective/Reflected/ Type-1 Cross Sited Scripting

In this attack social engineering is involved for the attack to be successful, the injected code is reflected off the web server in an error message, search result, or any other response that is sent to the server as part of the request. Reflected attacks are delivered to victims via routes such as an e-mail message or on some other web server. When a user is tricked into clicking on a malicious link the injected code travels to the vulnerable web server which is reflected back to the user's browser. The browser executes the code and targets vulnerabilities that occur in websites when data submitted by the client is immediately processed by server to generate results that are sent back to the browser on the client system. [14][15]

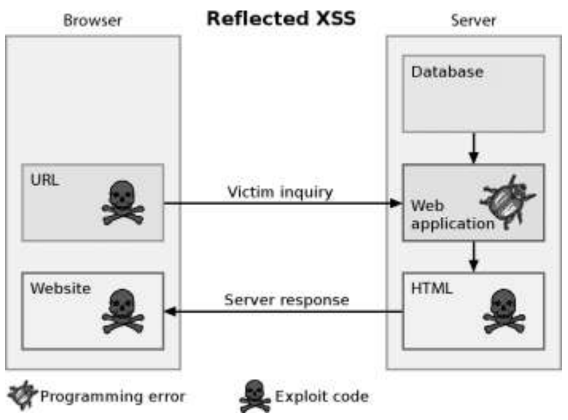

Figure 4: Reflected XSS attack

DOM-based/Local/ Type-0 XSS

These attacks occur in the content processing performed in client-side JavaScript. It exploits and targets vulnerabilities within the code of a webpage itself. Opening another Web page with malicious JavaScript code alters the code in the first page on the local system. In a local cross-site scripting exploit no malicious code is sent to the server rather they are interpreted by the browser to behave as they carried the malicious payload to the client from the server. [14][15]

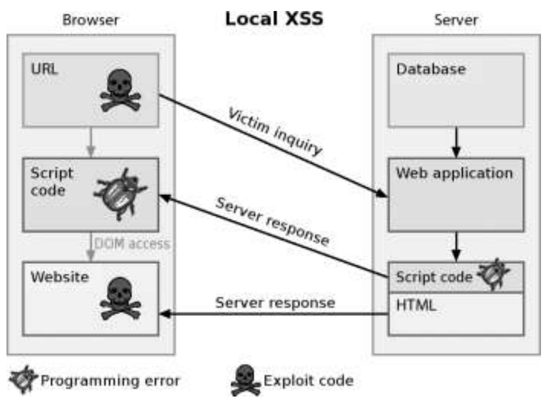

Figure 5: Local XSS attacks

XSS attack Scenario

When any Website becomes vulnerable an attacker can formulate XSS attack by using like injecting JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system with the victim's privileges. Attacker can account hijack, change user settings, perform cookie theft and poisonings. Cross site scripting (XSS) attack Scenario is presented below.

Attack via e-mail

The attacker sends an e-mail message to a victim containing malicious scripted link. When a user clicks on this link, the URL is sent to legitimate Site including the malicious code. The legitimate server sends a page back to the user; the malicious code will be executed on the client Web browser as shown in Fig. [8][9]

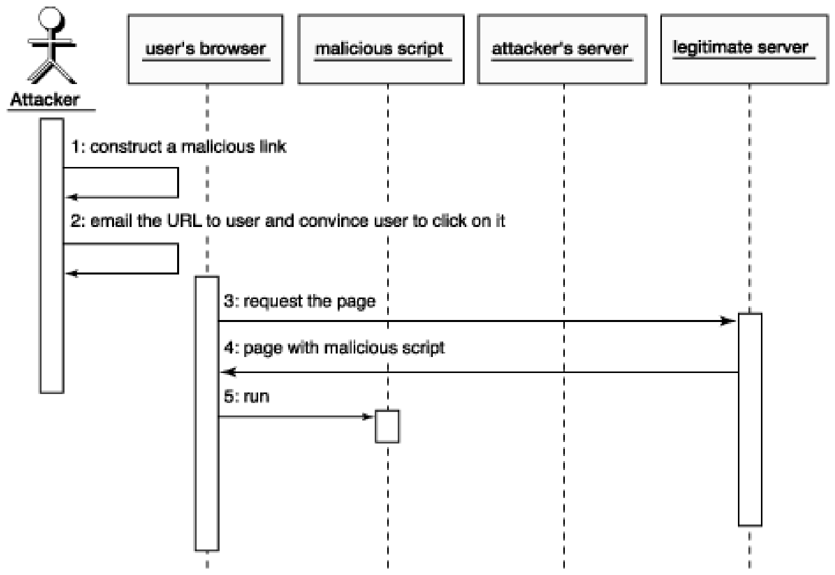
Attacker
user's browser
malicious script
attacker's server
legitimate server
1: construct a malicious link
2: email the URL to user and convince user to click on it
3: request the page
4: page with malicious script
5: run

Figure 6: Attack via e-mail

Cookie theft and account hijacking

The attacker files a page with malicious script to the part of the site that is vulnerable. When the page is displayed, the malicious script runs on it, collects the users' computer cookies, and sends a request to the attacker's Web site. Cookies are arbitrary pieces of data that are chosen by the Web server to send to the browser.

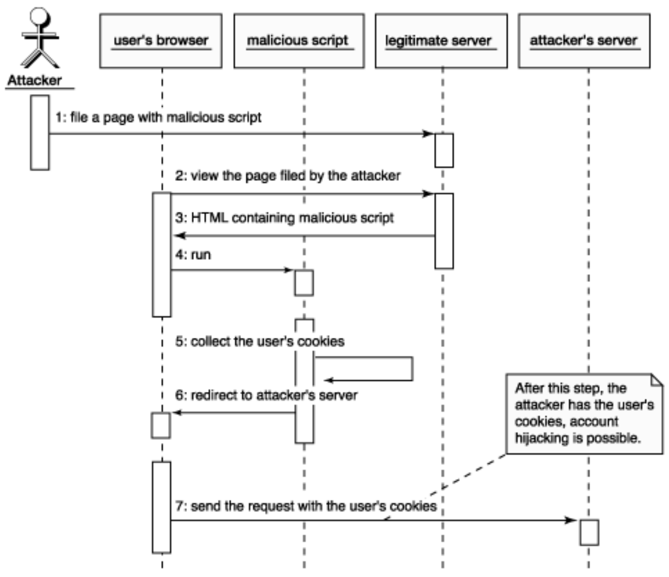
Attacker
user's browser
malicious script
legitimate server
attacker's server
1: file a page with malicious script
2: view the page filed by the attacker
3: HTML containing malicious script
4: run
5: collect the user's cookies
6: redirect to attacker's server
After this step, the attacker has the user's cookies, account hijacking is possible.
7: send the request with the user's cookies

Figure 7: Cookie theft and account hijacking

Using this technique, the attacker can gain sensitive data such as passwords, credit card numbers, and other information the user inputs. [8][9]

Sending an unauthorized request

In this scenario, the user unknowingly executes scripts written by attacker when they follow a malicious link in a mail message. The malicious scripts appears as if it is originated from the legitimate server in which the attacker has full access to all the document retrieved and may send data back to their site.

International Journal of Computers & Technology
Volume 4 No. 2, March-April, 2013, ISSN 2277-3061

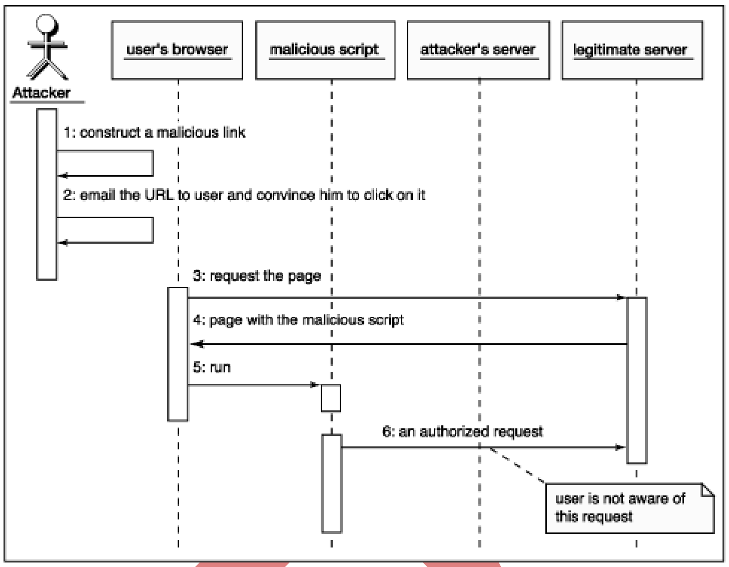
Attacker
user's browser
malicious script
attacker's server
legitimate server
1: construct a malicious link
2: email the URL to user and convince him to click on it
3: request the page
4: page with the malicious script
5: run
6: an authorized request
user is not aware of this request

Figure 8: Sending an unauthorized request

Attacker develops and exploits that posted data to a different page on the legitimate Web server. [8][9]

XSS DEFENDING

XSS defences can be broadly classified into four types: defensive coding practices, XSS testing, vulnerability detection, and runtime attack prevention.

Defensive coding

The best way to eliminate XSS vulnerabilities is to use defensive coding practices that validate and sanitize inputs to ensures that user inputs conform to a required input format. Replacement methods search for known bad characters and replace them with non-malicious characters and removal methods removes them. Escaping methods search for characters with special meanings for client-side interpreters and remove meanings. Restriction techniques limit inputs to some known good inputs.There are three classes of XSS defense that are emerging which include, Mozilla's Content Security Policy, Javascript Sandbox tools, and Auto-escaping templates.

Xss Testing

Input validation testing could uncover XSS vulnerabilities in Web applications. Only test cases containing adequate XSS attack vectors can induce original and mutated programs to behave differently. Testing involves detection of where safe encodings were not applied to emitted user-inputs, detecting where Unicode character transformations might bypass security filters and Detecting where non-shortest UTF-8 encodings might bypass security filters

Vulnerability detection

Other XSS defences focus on identifying vulnerabilities in server-side scripts. Static-analysis-based approaches can prove the absence of vulnerabilities.

- ***Static analysis***: These techniques identify tainted inputs accessed from external data sources, track the flow of tainted data, and check if any reached sinks such as SQL statements and HTML output statements.
- ***Static string analysis***: The enhancement provides more accuracy as it can analyse string operations' effects on inputs. However, when conducting static string analysis, it is difficult to model complex operations such as string-numeric interaction; thus, this approach can result in false positives if analysts make conservative approximations when handling such operations. Static string analysis also suffers from the limitations of blacklist comparisons.

- ***Combined static and dynamic analysis*****:** This method first identify the potentially faulty sanitization methods, then simulates the identified methods with a set of test inputs that contain attack strings and checks if any attack could still reach the sinks. [6]

Runtime attack prevention

The final group of XSS defences focus on preventing real-time attacks using intrusion detection systems or runtime monitors, which can be deployed on either the server side or client side. In general, these methods set up a proxy between the client and server to intercept incoming or outgoing HTTP traffic. The proxy then checks the HTTP data for illegal scripts or verifies the resulting URL connections against security policies.

Defending XSS vulnerability From User Side (Client Side)

It is possible to secure a site against a CSS attack in three ways:

- ***Input filtering***: By performing input filtering /input advanced filtering against HTML tags including JavaScript code should be applied, constrain inputs to their expected format and to render any dangerous input harmless by removing dangerous characters. A filter removes the characters <, >, and prevent XSS attacks when the content is inserted into a page in the context of an HTML element.

- ***Output filtering***: Output encoding refers to rewriting data such that it does not break out of the structural context into which it is inserted. It filters the user data when it is sent back to the browser, rather than when it is received by a script, encoding the characters **<**, **>** and **"** into their HTML entity equivalents: &lt;, &gt;, and &quot.
- ***By installing third party application firewall***: It intercepts CSS attacks reaching the web server vulnerable scripts, and blocks them. The application firewall inspects the data against various HTML tag patterns and JavaScript patterns, and if any match, the request is rejected and the malicious input does not arrive to the server. [8]

Defending attacks at Server side

Following approaches can be used on Server side to prevent XSS attacks.

- ***Encoding*****:** Server side encoding is a process where dynamic content will go through an encoding function where scripting tags will be replaced with codes in the chosen character set.
- ***The 1-2-3 of a sample encoding*****:** It is a way for a Web server to ensure that the generated pages are properly encoded is to pass each character in the dynamic content through an encoding function where the scripting tags in the dynamic content are replaced with codes in the chosen character set. This task is perfect for a custom tag library.
- ***Custom tag library*****:** A custom tag library is comprised of one or more Java language classes and an XML tag library description file which dictates the new tag names and valid attributes for those tags A custom tag library provides an architecture that is more flexible than a Java bean at encapsulating a complex operation.
- *Tag library descriptor*: A tag library descriptor is an XML file whose elements describe a particular tag library. The tag element defines the encode action, including an attribute, property. The tag class element defines the tag handler class Encode Tag.

CONCLUSION

XSS attacks are top threats to Web Security. Cross-site attacks depend on the trust developed between site and user. The severity of these attacks depends on the sensitivity of the data handled by the vulnerable site. Any site dealing with confidential or private user data should take necessary precautions to ensure applications remain protected. Preventing

XSS attacks requires diligence from the part of the programmers and the necessary security testing. We have presented Ways in which sites can be prevented from XSS attacks for client and server side.

ACKNOWLEDGMENTS

The authors would like to thank NIT Jalandhar, Infosys Limited and Chitkara University for supporting this research work.

REFERENCES

1. http://www.acunetix.com/blog/web-security-zone/articles/preventing-xss-attacks/
2. The Web Hacking Incidents Database
3. Gary McGraw, Software Security: Building Security In, Addison-Wesley Professional, 2006.
4. Michael Howard, David LeBlanc, and John Viega, 19 Deadly Sins of Software Security –
5. S. Fogie et al., XSS Attacks: Cross Site Scripting Exploits and Defense, Syngress, 2007.
6. N. Li et al., "Perturbation-Based User-Input-Validation Testing of Web Applications," J. Systems and Software, Nov. 2010, pp. 2263-2274.
7. D. Balzarotti et al., "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," Proc. 29th IEEE Symp. Security and Privacy (SP 08), IEEE CS, 2008, pp. 387-401.
8. A. Kiezun et al., "Automatic Creation of SQL Injection and Cross-Site Scripting Attacks," Proc. 31st Int'l Conf. Software Eng. (ICSE 09), IEEE CS, 2009, pp. 199-209.
9. M.T. Louw and V.N. Venkatakrishnan, "Blueprint: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers," Proc. 30th IEEE Symp. Security and Privacy (SP 09), IEEE CS, 2009, pp. 331-346.
10. E. Kirda et al., "Client-Side Cross-Site Scripting Protection," Computers & Security, Oct. 2009, pp. 592-604.
11. P. Saxena, D. Molnar, and B. Livshits, SCRIPTGARD:
12. M. Louw and V. Venkatakrishnan, "BLUEPRINT: Robust Prevention of Cross-Site Scripting Attacks of Existing Browsers," IEEE S&P, Oakland, May 2009, pp. 331-346.
13. B. Livshits and U. Erlingsson, "Using Web Application Construction Frameworks to Protect Against Code Injection Attacks," Proc. of PLAS, California, 2007, pp. 95-104.
14. P. Vogt et al., "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," Proc. of the NDSS, California, Feb. 2007.
15. P. Bisht and V. Venkatakrishnan, "XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks," Proc.of the 5th DIMVA, Paris, July, 2008, pp.23-43.