Cross Site Scripting Vulnerabilities and Defences: A Review

Swati Maurya¹, Anita Singhrova²
M.Tech Student¹, Professor²

Abstract

With the advancement in the internet technology since last two decades, the dependence on web applications has increased rapidly. All the facilities are nowadays available online at the ease of just one click. As a result, Web applications are prone to cyber-attacks which has major consequences such as theft of personal secure data and information tampering by 'Cookie stealing' or 'Session Hijacking'. Cross site scripting is one of the most common application layer web attacks and security vulnerability. This paper provides review of the concept and the approaches provided for vulnerability detection, defence mechanism and security analysis.

Keywords— XSS; Web Application Vulnerabilities; Cross-site Scripting; Security

1. Introduction

A vulnerability is a weakness or loophole in the application that can be misused to hamper the security of the system/application. Web application vulnerabilities have severe impact than other vulnerabilities. The failure in properly handling user inputs in a web application leads to Web application vulnerabilities. If the attackers find any of such flaw in the application vulnerability, they can exploit it to a cyber-crime. Web applications are prone to cyber-attacks as these attacks are launched on port 80 that remains open. Defence mechanisms through SSL and firewalls are not effective for Application level attacks as they do not provide coverage for port 80 attacks [1].

Gartner Security states that the application layer currently contains 90% of all vulnerabilities [2]. The list for Top 10 Web application security flaws by OWASP is shown in Table I.

1.1 HTTP and cookie concept

Hyper Text Transfer Protocol (HTTP) is an Internet Client/Server protocol which provides fast and reliable transfer of hypertext data. It is a stateless protocol that do not support sessions and no binding is done for any request and response with the corresponding client and Cookies are used.

In case of web applications vulnerabilities, the attackers need only a web browser to gain access to the web application and misuse the secure sensitive information. It provides a platform for the launch of severe attacks on the local systems of web users.

Cross-Site Scripting (XSS) is one of the most common application layer web attacks and security vulnerability which has major consequences such as theft of secure data and information tampering. XSS is the class of web application vulnerabilities which target scripts embedded in a web page to be executed either on client side or server side which will be manipulated by malicious user in the desired manner. XSS occurs when web applications take input from users and dynamically insert them in web pages without validating it. The concept of XSS is illustrated in Figure 1.

Table 1: Web Application Security Flaws [3]

<table>
<thead>
<tr>
<th>OWASP Top 10 list for Application Security- 2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 Injection</td>
</tr>
<tr>
<td>A2 Broken Authentication and Session Management</td>
</tr>
<tr>
<td>A3 Cross Site Scripting (XSS)</td>
</tr>
<tr>
<td>A4 Insecure Direct Object References</td>
</tr>
<tr>
<td>A5 Security Misconfiguration</td>
</tr>
<tr>
<td>A6 Insecure Deserialization</td>
</tr>
<tr>
<td>A7 Missing Function Level Access Control</td>
</tr>
<tr>
<td>A8 Cross Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>A9 Using Known Vulnerable Components</td>
</tr>
<tr>
<td>A10 Invalidated Redirects and Forwards</td>
</tr>
</tbody>
</table>

In case of web applications vulnerabilities, the attackers need only a web browser to gain access to the web application and misuse the secure sensitive information. It provides a platform for the launch of severe attacks on the local systems of web users.

Cross-Site Scripting (XSS) is one of the most common application layer web attacks and security vulnerability which has major consequences such as theft of secure data and information tampering. XSS is the class of web application vulnerabilities which target scripts embedded in a web page to be executed either on client side or server side which will be manipulated by malicious user in the desired manner. XSS occurs when web applications take input from users and dynamically insert them in web pages without validating it. The concept of XSS is illustrated in Figure 1.
as a solution for this problem which stores and retrieves information from client side.

Since the Cookies store information in a format that is easily readable, the information is vulnerable to data breaching and can be manipulated. Thus, to reduce the risk associated, the information exchange should occur through secure channel (SSL). Then also the information is not secure fully and can be stolen by any malicious content (Trojans). Cross site scripting is most occurring attack in these scenarios. [4]

To summarize, it can be stated that, XSS is a web application vulnerability that is utilized by any malicious user/attacker to bypass the cookie protection mechanisms.

1.2 XSS

Cross-Site Scripting is a web vulnerability in which web applications blindly trust the data received from the client. The most common example is the insertion of executable scripts in the URL of a website. XSS is used to attack the websites that manage Cookie based sessions and try to steal secure information from them.

In XSS, the malicious script is present in the hyperlinks which are used to gather data. The adversaries attract users to click on the links in websites or through emails and the infected code contains malicious script that will be forwarded to web application server in the form of Request message. As a response from server, the output page is generated and contains infected code which will appear as a valid content and hence the malicious script will be executed by the web application of the client. Thus, the servers mistakenly act as a launching platform for running the malicious script in the browsers through dynamic output pages. Though the servers with static output page are not vulnerable to such attacks as they do not get manipulated in terms of web page manipulation. In such a scenario, the attacker will try to insert a new executable malicious code to be run by the browser. Irrespective of the technique or mechanism used, the ultimate concern of the attacker is to target client information through XSS attack and steal secure information. The above described concept is summarized in Figure 2.

Figure 1: Summary of XSS attack [4]

Since the Cookies store information in a format that is easily readable, the information is vulnerable to data breaching and can be manipulated. Thus, to reduce the risk associated, the information exchange should occur through secure channel (SSL). Then also the information is not secure fully and can be stolen by any malicious content (Trojans). Cross site scripting is most occurring attack in these scenarios. [4]

To summarize, it can be stated that, XSS is a web application vulnerability that is utilized by any malicious user/attacker to bypass the cookie protection mechanisms.

1.2 XSS

Cross-Site Scripting is a web vulnerability in which web applications blindly trust the data received from the client. The most common example is the insertion of executable scripts in the URL of a website. XSS is used to attack the websites that manage Cookie based sessions and try to steal secure information from them.

In XSS, the malicious script is present in the hyperlinks which are used to gather data. The adversaries attract users to click on the links in websites or through emails and the infected code contains malicious script that will be forwarded to web application server in the form of Request message. As a response from server, the output page is generated and contains infected code which will appear as a valid content and hence the malicious script will be executed by the web application of the client. Thus, the servers mistakenly act as a launching platform for running the malicious script in the browsers through dynamic output pages. Though the servers with static output page are not vulnerable to such attacks as they do not get manipulated in terms of web page manipulation. In such a scenario, the attacker will try to insert a new executable malicious code to be run by the browser. Irrespective of the technique or mechanism used, the ultimate concern of the attacker is to target client information through XSS attack and steal secure information. The above described concept is summarized in Figure 2.

Figure 1: Summary of XSS attack [4]

Apart from cookie stealing, XSS attack can do other security violations such as hacking account, data breaching, manipulating content of a web site and denial of service. [6]

Thus as a general measure to overcome XSS, validate and filter all user inputs. For implementing this, there are two approaches possible:

- Creation of ‘Blacklist’ to filter out unwanted user inputs.
- Creation of ‘Whitelist’ to specify expected user inputs.

2. Classification Of XSS

XSS can be classified broadly into three categories on the basis of manipulation of user inputs:

2.1 Stored XSS Vulnerability (Persistent XSS): When user inputs are stored in database first and then accessed and used for responses. e.g: Forums and Social networking sites.

2.2 Reflected XSS Vulnerability (Non persistent XSS): When user’s input information is used in immediate web page response without checking for validity.

2.3 DOM-based Vulnerability: When invalidated user input is used by client side program that was obtained dynamically from Document object model(DOM) structure. In this XSS attack, the attack payload is executed by modifying the DOM environment at client side in victim's
3. Classification Of Vulnerability Detection Approaches

The vulnerability detection approach is classified on the basis of analytical approach into the following categories:[7]

3.1 Static analysis approaches[7]

Static analysis approaches are used for finding errors in the early phases of development and before the programs is run for the first time and hence it reduces cost for after checking. The recent researches in Static Analysis Algorithms are based on analysing the program by reading the code and its constructs and then analysing the abstract model. The techniques that are used for analysing the programs to trade off precision are: Flow sensitive analysis or Flow insensitive analysis, Interprocedural or Intraprocedural analysis, Path sensitive or Path insensitive analysis.

3.2 Dynamic Analysis Approaches[7]

Dynamic analysis approaches are better than Static analysis as they analyse the information obtained during program execution to detect vulnerabilities and do not generate false positive and false negative results. It is usually performed during testing phase of SDLC or runtime after software release.

3.3 Hybrid Analysis Approaches[7]

Hybrid approaches are based on the combination of static and dynamic analysis approaches. In these approaches dynamic analysis techniques improve the false alarms of static analysis approaches and provide precise results.

4. XSS Defence Mechanism

As a preventive measure for the script getting executed through the inputs given by user, there are two possible approaches:

4.1 Content Filtering: Detection and removal of scripts from untrusted HTML before sending it to the browser.[8]

4.2 Browser Collaboration: Browser’s defences implement client side filters that include the authorised scripts to be allowed for execution at client. [8]

The summary of various XSS defence mechanisms are presented in Table 2:

<table>
<thead>
<tr>
<th>Author</th>
<th>Implementation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shanmugam et al (2007)</td>
<td>Server side solution</td>
<td>The proposed solution, Service oriented architecture blocks XSS attacks on web applications irrespective of the language and platform of development by using XML and XSD for interoperations of the independent services which validate the inputs submitted by user through XML documents.</td>
</tr>
<tr>
<td>Keyzun et al (2009)</td>
<td>Server side solution (Dynamic taint analysis)</td>
<td>Signature based misuse detection approach utilises negative security model to reduce the false positives.</td>
</tr>
<tr>
<td>Shahriar et al (2011)</td>
<td>Server side solution</td>
<td>Attack creation technique that combines Input generation, Taint propagation and Input Mutation for generating concrete inputs (attack vectors) which ensure that data flows from input to sensitive sink.</td>
</tr>
<tr>
<td>Shar et al (2011) [13]</td>
<td>Server side solution (Static Analysis)</td>
<td>Code auditing approach extracted the defence features in the form of Tainted Information flow graph and rules provided classify the vulnerable HTML output into Vulnerable, Not Vulnerable, Probably vulnerable or Probably not vulnerable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proxy)</th>
<th>firewal that monitors incoming connections and disallows requests if it is from an unauthorized user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iha et al (2009) [16]</td>
<td>Client Side Solution (Web Browser solution)</td>
</tr>
<tr>
<td>Shahriar et al (2013) [17]</td>
<td>Client side solution (Web proxy)</td>
</tr>
<tr>
<td>Avancini et al (2011) [18]</td>
<td>Security Testing</td>
</tr>
<tr>
<td>Ruse et al (2013) [20]</td>
<td>Security Testing (Dynamic)</td>
</tr>
<tr>
<td>Salas et al (2014) [21]</td>
<td>Security Testing</td>
</tr>
<tr>
<td>Rocha et al (2014) [22]</td>
<td>Security Testing</td>
</tr>
</tbody>
</table>

5. Conclusion

Despite the numerous solutions provided so far, Cross site scripting still remains one of the major concern for the web applications. No single solution is available that could prove to be effective in mitigating XSS attacks. Further research to provide a complete solution for vulnerability removal from the source code of applications before deployment is needed.

6. References


