Prevention to Cross-site Scripting Attacks: A Survey

Manisha S. Mahindrakar

Assistant Professor, Computer Science and Engineering Department,
Shri Guru Gobind Singhji Institute of Engineering and Technology, Nanded, Maharashtra, India

Abstract: ***With Internet becoming ubiquitous in every aspect of our life, there is an increase in the web applications providing day to day services like banking, shopping, mailing services, news updates, etc. But most of these applications have vulnerabilities or security loopholes like Cross site scripting (XSS), Cross-site request forgery (CSRF), SQL Injection which are being exploited by the hackers for malicious purposes. Hence there is a need for API's/automated security tools to identify and/or prevent these vulnerabilities before the application goes live. This survey paper focuses on various security tools and prevention methodologies available to mitigate attacks due to Cross-site Scripting (XSS) and Cross-site request forgery (CSRF) vulnerabilities.***

Keywords: Web Applications, Cross-site Scripting (XSS), Cross-site Request forgery (CSRF/XSRF).

1. Introduction

With the proliferation of the Internet, there has been a surge in the web services being offered by many corporations like e-banking, e-shopping, etc. As most of these applications are not developed with best security practices, there is an increase in the malicious attacks against these services, which exploits the vulnerabilities in these applications to acquire material gains or to steal the credentials of the novice users who use these web services. This has resulted in more research focus in this domain to create new tools and techniques to subvert these kinds of attacks. There are many research groups in academics and industry working in this domain to find out more secure programming practices and tools to identify the vulnerability of these applications during development phase and attacks during the real time.

The OWASP Top 10 report [1] lists the following as the ten most critical web application security vulnerabilities that are been exploited:

- Cross Site Scripting (XSS)
- Injection Flaws (SQL Injection, XPath Injection, LDAP Injection etc)
- Malicious File Execution
- Insecure Direct Object Reference
- Cross Site Request Forgery (CSRF)
- Information Leakage and Improper Error Handling
- Broken Authentication and Session Management
- Insecure Cryptographic Storage
- Insecure Communications
- Failure to Restrict URL Access

In this paper, I have concentrated on Cross-site Scripting (XSS) vulnerability, which facilitates the hacker to insert some malicious script to the web application that may cause any kind of harm to legitimate user.

Web applications have evolved from a static medium that has user interaction limited to navigation between web pages, to a highly interactive medium serving up personalized content. Web language such as HTML has capability to support dynamic data execution of web applications required to serve personalized contents. HTML allows inline constructs both to embed untrusted data and to invoke code in high order languages such as JavaScript.

Due to this ad-hac evolution to support demands of the growing web, HTML and other web languages lack the principled mechanism to separate untrusted data (user contents) from trusted data. As a result, there are cross-site scripting attacks on web applications. To mitigate problem of XSS attacks, XSS defense is required. There are various XSS defenses categorized as shown in Fig 1.

This paper organized as follows. Section II discuss about various XSS defenses and Section III discuss advantages and disadvantages of each defense.

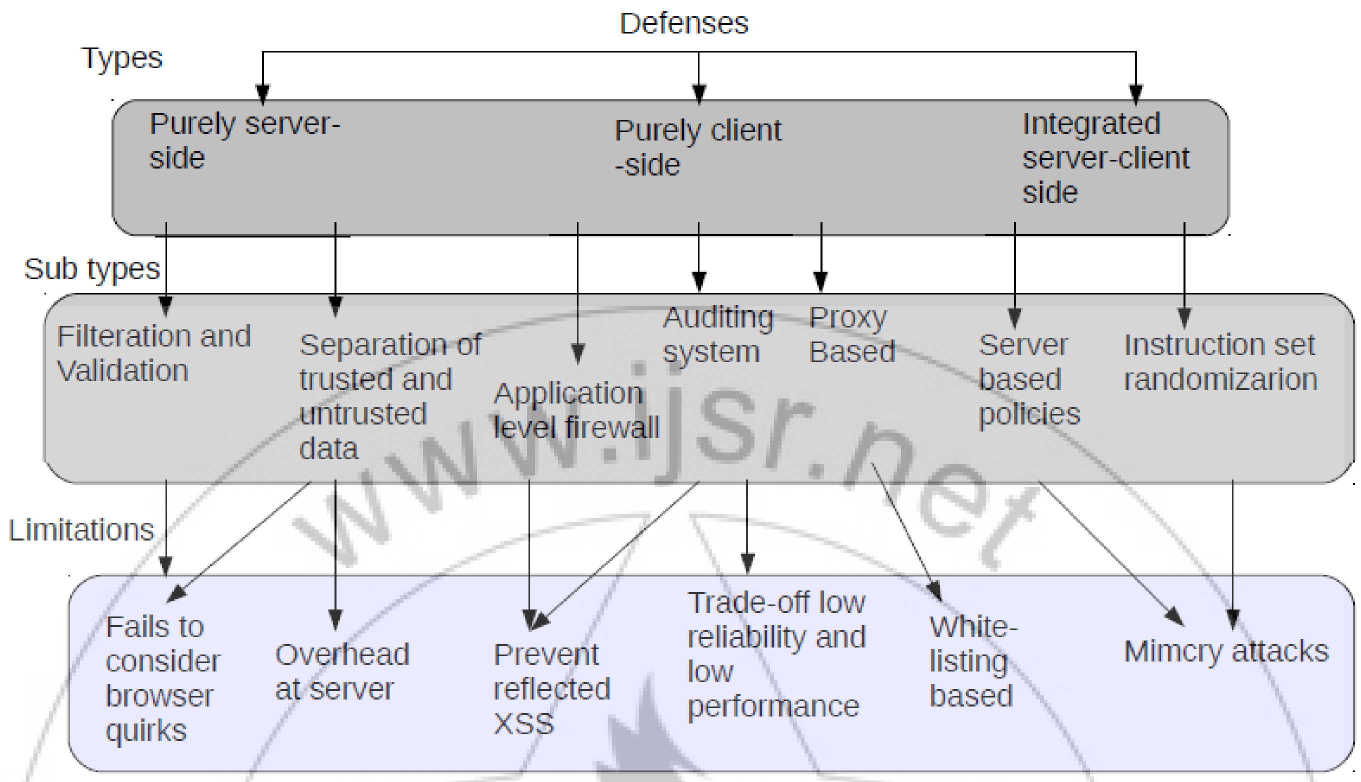
Defenses
Types
Purely server-side
Purely client -side
Integrated server-client side
Sub types
Filteration and Validation
Separation of trusted and untrusted data
Application level firewall
Auditing system
Proxy Based
Server based policies
Instruction set randomizarion
Limitations
Fails to consider browser quirks
Overhead at server
Prevent reflected XSS
Trade-off low reliability and low performance
White-listing based
Mimcry attacks

Figure 1: Types of Existing Defenses

2. Existing Defenses

XSS defences are categorized as purely server-side, purely client-side and integrated server-client side.

2.1 Server-side Defences

Basic and primary defence for XSS is server-side validation and filtration of the contents generated by user. Common problem with all these mechanisms is the assumption that parsing and rendering on client browser is consistent. Some of the filtering mechanisms are as explained below.

Filtering:

Now days, Web applications need mechanisms to provide users with ability to format user data (profile or comments or blogs) with rich text ie using HTML/CSS. To provide this functionality, developer allows user to use intermediate language to format their posting. Many lightweight markup languages are available like BBCode[2], Wikitext [3] and Textile [4]. These languages are parsed and translated to markup language as web browser understands only HTML/XHTML.

For example:

BBcode: [url=http://www.cse.iitb.ac.in]CSE[/url]
This gets translated in HTML to render in web browser.
HTML/CSS:<aref=”http://www.cse.iitb.ac.in”>CSE</a>
Use of such intermediate options is safest option to allow user the subset of HTML functionality.

The other option is to allow user to input HTML/CSS directly to format their data. But user input cannot be trusted, so web application should be able to detect and remove malicious code in user's data if any. There are few solutions developed to detect and remove malicious code.

Following are the few solutions:

Striptags():
The striptags() is PHP function which is used to cleanup HTML. It is worst to detect and remove XSS attacks as it could not validate attributes of tag. Attribute's value can be used to insert malicious code for exploiting web application. Validation of attributes can be achieved with a series of regular expressions that strip out on event. But still web application may remain vulnerable to XSS due to quirky browser behaviour. [5]

HTML_safe:
HTML_Safe mechanism involves parsing HTML with a SAX parser and performing validation and filtering depending on the handlers called. strip_tags can only strip tags. HTML_safe strips down all active content, including tags, attributes and values of attributes. [6][9]. This parser strips down all potentially dangerous content within HTML like:

- Opening tag without its closing tag
- Closing tag without its opening tag
- It also has blacklisted some of the tags, attributes and protocols like javascript:,vbscript:,about:
- Active contents in style tags

Kses: Kses is an HTML/XHTML filter written in PHP. It removes all unwanted HTML elements and attributes and it also does several checks on attribute values. Kses helps to avoid XSS.[7]

Kses uses set of API's to allow users to configure the filter to add and remove protocols, tags and also its attributes. Users are supposed to be very cautious in using API's, as different ways of using API's results in different functionality.

HTMLawed: The set of features provided by htmLawed are appreciable as it can auto-correct and beautify HTML markup and restrict HTML elements (tags), attributes and URL protocols in the input. It also balances tags and checks for proper nesting of HTML elements. HtmLawed is highly customizable. This filter is very useful to make text with HTML more secure. This filter removes all occurrences of scripts. It does not validate script to check its vulnerability for XSS.[8]

HTML purifier: It is a standards-compliant HTML filter library written in PHP. Developers of HTML Purifier claims that it will remove all scripting code by auditing it thoroughly, which is the loss of functionality provided. It is big code to fit in one file and has huge includes list. Due to big code it takes more time to give results. That means it is slow as compared to HTMLawed. [10]

Filtering and validation cannot detect XSS attacks which are commingled with trusted code. Solutions like XSS Guard [13] and Blueprint tries to prevent XSS attacks by dynamically learning the set of scripts that a web application intends to create, for any HTML request. It then removes any script from the output which is not intended by the web application. Problem with these types of defenses is that they have to consider all browser quirks because browser is the one who detects and processes the scripts and render the web page.

2.2 Client-side Defences

I have further sub-categorized Client-side defences as Application-level firewall, auditing system and Proxy based.

Application-level Firewall:

These types of firewalls (filters) mitigate XSS attacks by preventing attacker's script leaking sensitive data to attacker's server. These filters monitor the flow of sensitive information within website and aims in blocking the script which tries to send information to the web site of different domain.

Auditing system:
These types of filters block injections typically by matching the contents of HTTP response with the contents of HTTP request that generated the response. These types of filters often get success in detecting reflected XSS attacks but fails in detecting stored XSS attacks.

Low Performance: The filter could re-implement exactly the same process as the browser, but such a filter would double the amount of time spent in parsing the HTTP response. For example, noXSS [11] contains an entire JavaScript parser. This type of filters gives perfect reliability by lowering the performance. Low performance is quite unacceptable for todays web sites as they contain lots of images and videos. Also these types of filters can easily get bypassed by using special characters and encoding methods in attack string.

Low Reliability: These filters approximate the process at browser by using set of regular expressions instead of simulating the complete process. These regular expressions work faster than complete HTML parser but it increases the number of false positives.

Proxy based:
Proxy based method involves sending a request to a server, keeping a copy of HTML tags on a client side proxy and then forwarding the request to the web server. On receipt of the response, the HTML tags are checked for the tags that were sent within the request. If any tags match, the response is marked as XSS vulnerable.

This method is advantageous as it is simple to implement and configure on the client side. There is little overhead because the checking of HTML is done on the client side.

Another web proxy based solution is to use white-listed domains. These solutions can use both manual and automatically generated rules to detect and mitigate XSS attacks. The rules are used to specify allowed and denied domains. Manual rules are the ones specified by user. User has to use wild cards to permit or deny requests matching the rules. A firewall prompt is another method to generate rules. According to this method, request generates prompts to allow or deny in case it does not match with any of the existing rules. In this type user can specify permanent or temporary rules. Temporary rules remain active for present session whereas permanent rules remain permanently in the policy file. Lastly user can specify a session for generating rules, in which tool generates rules based on domains visited by user during that session.

2.3 Integrated Server-Client Defenses:

Assumption in this approach is that web application knows what the legitimate scripts are. So web application helps the client side (browser) to isolate the legitimate contents from user (untrusted) contents either by using policy or randomizing the code.

Based on server policies:

In this kind of defenses, Web application embeds a policy in its pages that specifies which scripts are allowed to execute. Whenever browser encounters a script while parsing the response, it enforces the embedded policy perfectly before execution of the script. One of the defense of this type is Browser enforced embedded policies (BEEP)[12]. This defense prevent script injection depending on two observations, first is Browser perform perfect script detection and second is web application developer knows exactly what scripts should be executed for application to function properly.

The advantage of this type of XSS defense is, they are easy to implement. Policies are flexible means can be changed whenever required. Browser need not to modify much to enforce the embedded policy.

Instruction set randomization: Instruction Set Randomization (ISR) is a technique to obscure the instruction set. In order for the injected code to have the intended effect, the attacker must know the instruction set of the target application. Hence, a general technique for defusing code injection attacks is to obscure the instruction set from the attacker. Instruction Set Randomization (ISR) is a technique for accomplishing this by randomly altering the instructions used by a host application. By changing the instruction set, ISR defuses all code injection attacks.

In Noncespaces[14], Web application randomizes XML tag prefixes before delivering a document to client. Due to randomization, it becomes hard for attacker to predict the prefixes, so injected attack fails. It is dom-based solution; it cannot prevent the attack, which target client-side scripting code, which processes user (untrusted) contents in unsafe manner during its execution.

Another approach of this type is xJS[15], features of this approach are as follows

- It collaborates between server-side and client-side.
- Know exactly what is intended by server.
- This solution can be applied to already existing web applications with minor changes as solution is implemented on apache web server.
- Apache web server is more popular to host web applications.
- Randomization of scripts is done at apache web server.
- Less overhead at server as lightweight XOR operation is used for randomization.

3. Advantages and Disadvantages

This section will discuss about advantages and disadvantages of each scheme.

By study and analysis of server-side XSS solutions, I can conclude that Filtering and validation is useful as first level of defense against XSS and limit XSS to certain level. Some of these solutions are really helpful to avoid XSS provided it either needs to configure (set many parameters) carefully (HTMLawed) or hamper the performance of web application. (HTML-Purifier) Absence of balanced approach which differentiates between malicious and non-malicious XSS attacks is matter of concern for security of web application.

One major difficulty with client-side defence ie firewalls (filters) is that, now-a-days many websites frequently export data to third party web sites. For example Modern web sites often have rich interaction with other web sites via advertising and gadgets. So there is need to differentiate between benign and malicious data. For this purpose, Filters use sophisticated analysis techniques like taint tracking and static analysis.
Other difficulty with this type of filters is, it fails to prevent attacks performed by breaking confidentiality of victim's session with target (vulnerable) web site. For example many web sites provide a user-to-user messaging facility. On such web sites attacker can send victim's confidential information to his or her own user account in a user-to-user message. Later attacker can login to these web sites to read or retrieve stolen information. Another example of such attacks is banking web site. Attacker can transfer money from victim's account to his or her own account on banking web site.

Proxy-based solutions are simple to implement and configure on the client side. There is little overhead because the checking of HTML is done on the client side.

The major disadvantage of this technique is that it is not very smart. It will mark any HTML that is returned that matches the request as XSS vulnerable, even though it is safe. A technique to fix this error is to apply a length constraint to tags checked, but this is still not a fool proof technique to prevent the incorrect XSS vulnerability indication.

The advantage of BEEP [12] XSS defense is, It is easy to implement. Policies are flexible means can be changed whenever required. Browser need not to modify much to enforce the embedded policy.

The main disadvantage of this kind of defenses is that they fail to apply web application's logic perfectly. As a result they easily get attacked with white-listed scripts. If the attack contains white listed scripts then browser enforced policy allows the script for execution and attack works successfully. These types of attacks are called mimicry attacks.

4. Future Scope

After studying all these prevention and detection tools, we can design Model API which will work very fine in stripping out the legitimate cross-site scripts (XSS), XSS worms and virus as well. Tool should be solution to all server side scripting languages like PHP, JSP and ASP.

References

[1] OWASP Top 10, The Ten Most Critical Web Application Security vulnerabilities, http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf, Last Accessed: July 7, 2009.
[2] BBCode, http://en.wikipedia.org/wiki/BBCode, Last Accessed: July 7, 2009.
[3] Wikitext, http://en.wikipedia.org/wiki/Wikitext, Last Accessed: July 7, 2009.
[4] Textile, http://textism.com/tools/textile/, Last Accessed: July 7, 2009.
[5] Strip_tags–Manual, http://php.net/manual/en/function.strip-tags.php, Last Accessed: July 8, 2009.
[6] HTML_Safe, http://pear.php.net/package/HTML_Safe/, Last Accessed: July 8, 2009.
[7] Kses, http://sourceforge.net/projects/kses/, Last Accessed: July 8, 2009.
[8] htmLawed: www.bioinformatics.org/phplabware/internal_utilities/htmLawed/index.php, Last Accessed: July 8, 2009.

[9] Safe_HTML_Checker, http://simonwillison.net/2003/Feb/23/safeHtmlChecker/, Last Accessed: July 8, 2009.
[10] HTML Purifier, http://htmlpurifier.org/, Last Accessed: July 8, 2009.
[11] addons.mozilla. https://addons.mozilla.org/en-us/firefox/addon/noxss/
[12] BEEP T. Jim, et al. WWW 2007, Tahoma R. Cox et al. S&P 2006, Browsershield C. Reis et al. OSDI 2006.
[13] Prithvi Bisht and V.N. Venkatakrishnan, XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks, Springer-Verlag Berlin, Heidelberg ©2008, pp. 23 - 43
[14] Matthew Van Gundy, Hao Chen, "Noncespaces: Using randomization to defeat cross-site scripting attacks", Elsevier Advanced Technology Publications Oxford, UK, UK, Volume 31 Issue 4, June, 2012, PP 612-628.
[15] Elias Athanasopoulos, Antonis Krithinakis, Spyros Ligouras, Evangelos P. Markatos, and Thomas Karagiannis. "xjs: Practical xss prevention for web application development", 2010

Author Profile

Manisha Mahindrakar received the B.E. and M. Tech. degrees in Computer Science and Engineering from SGGSIET, Nanded, MH 2001 and Indian Institute of Technology, Bombay, MH in 2011 respectively. She is working as Assistant professor in computer Science and Engineering Department form 2004 till date.