Study of Cross-Site Scripting Attack, Detection and Prevention

Author1, Apoorv Anupam, III yr B.Tech CSE, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai
Email:apoorv.anupam2018@vitstudent.ac.in

Author2, Saurabh Mohata, III yr B.Tech CSE, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai
Email:saurabh.pawankumar2018@vitstudent.ac.in

Author3, Ayush Anupam Gupta, III yr B.Tech CSE, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai
Email:ayushanupam.gupta2018@vitstudent.ac.in

Author4, Dr. T.Subbulakshmi, Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai
Email:research.subbulakshmi@gmail.com

Abstract

Cross-site Scripting vulnerability abbreviated as XSS, unlike most vulnerabilities that influence server-side, is a vulnerability that emerges on the client-side. Cross-Site Scripting attack is one of such attacks against the web applications and servers in which a client has to trade off its web browser's resources (e.g. cookies, passwords, etc.). Badly coded WebApplication creates vulnerabilities that can set a base access to inject malicious script into it. In this paper, we described the attack generation, detection, and preventive measures of the Cross-Site Scripting (XSS) vulnerability. We look into the working of XSS attack along with the vulnerabilities it exploits, its detection using generic vulnerability scanner. We discussed better coding practice and how the XSS attack can be prevented. We also look into how attackers steal information by injecting the malicious script on web applications and tested the vulnerability scanner on the Damn Vulnerable Web Application (DVWA) which is *used to learn and test web penetration skills and is a good platform to showcase the XSS attacks. This analysis of Cross-Site Scripting Attack (XSS) could help to avoid the attack.*

Keywords: Cross-Site Scripting Attack, Web application, Cookies

1. Introduction

Cross-site scripting attack gets executed on the client side of the web apps by the injection of malicious code. The client side of a web application is usually the software that is used to interact with the web application and in most cases, it is a browser that is used to interact with the web application. So, in cross-site scripting attack, the attacker injects malicious code on to the web browser to make the web application do something that it is ideally not supposed to do. Depending on what kind of cross-site scripting is being used the malicious script executes when the victim visits the web page, it could be a single web page or maybe the web server. Each time a user wants to send a request to the web server, it has to include the corresponding cookie in its request, so that the webserver associates the cookie to the corresponding user. And thus, the cookies become targets for the attackers. It can be thought of as a client-side code infusion that happens, using JavaScript, HTML, VBScript, etc. This attack is mainly used to steal sensitive information like cookies, session, tokens and other sensitive information. Cross-site scripting Attacks (XSS) can also be used to modify the contents of the website.

1.1. Cookies

A cookie is a little bit of information stored on our Computer by the internet browser while browsing a site. Cookies were intended to be a reliable system for sites to recollect data or to record the browsing activity. Cookies do have some technical downsides specifically they don't recognize the user in every case precisely, and additionally can be utilized for security attacks which could be a major protection concern.

1.2. Types of cross-site scripting attack

Cross-site Scripting can be characterized in the following ways:

1 - Stored XSS

2 - Reflected XSS

3 - DOM-based XSS

While there are a number of variations in XSS Attacks, all cases follow a pattern. The goal is to make a victim coincidentally execute a maliciously injected script. An attacker's malicious script is often called a payload.

Stored XSS happens when client provided input is stored on a web application and afterward delivered inside a page. An attacker typically abuses this weakness by infusing XSS payloads. Stored XSS is also called persistent XSS.

Reflected XSS attacks permit an attacker to fool a victim to click on a XSS payload. In this case the data is not stored on the web server. When the payload is sent to a reflected XSS vulnerable site page, the payload is reflected back as a feature of the HTTP response.

DOM Based XSS which makes use of the document object model to inject the malicious script. The payload is executed because of adjusting the DOM condition in the victim's web browser and also the page wouldn't change but the client-side code gets executed because of that adjustment.

2. Related Work

There is a lot of work that has been done in the area of XSS attacks, detection and prevention. With xss attacks having the possibility of stealing important and confidential data, a lot of research has been done in this area. From Detection methods implemented at server side to new ways of making the web application code secure[4]. Some of the work is showcased here.

There are a variety of prevention and detection techniques to prevent XSS vulnerabilities. Preventive approaches consist of secure programming guides to inform the developers to use encoding features correctly. Some of the well-known guides like the OWASP XSS cheat sheet [10] and best practices by Graff and Wyk [12]. Also, there are some filters, e.g. [5], that try to block redundant JavaScript programs in HTML body context. Such filters are checked to verify as many patches have been issued.

Ismail et al. [7] described an XSS detection mechanism through the usage of a server-side proxy, incoming parameters are checked for contained HTML mark-up. If such a parameter may be identified, then the respective HTTP response is tested if the identical HTML mark-up can be determined.

A brilliant suggestion to counteract the XSS attack from the server side is to use the validation or filter method. Filtering would be a good initial guard against xss but would quickly be ineffective when it comes to several instances of attacks. [2]. Badly written PHP can also increase the vulnerability of the system Naturally the web browsers have filters to protect against XSS attacks, but with badly written PHP code, these filters can be bypassed. [3] . XSS attacks also create a lot of problems for the Online Social Networks. A good implementation of a framework based on request authentication and view isolation approach could help make the networks much more secure. Validating a string at vulnerable checkpoints to detect XSS attack is a good filtering approach. [1]. Another good way of dealing with XSS attacks is with keenly checking the cookies and session variables. [9]

3. Generation of XSS Attack

Cross Site Scripting Attacks (XSS) attacks are those attacks against the web applications which are used to steal the cookies from a web browser‘s database. It mostly consists of three useful commodities i.e. Attack Server, Victim Server and Web Application.

After finding if the corresponding web application is vulnerable to Cross-Site Scripting attack, the attacker will post a malicious Code on that Vulnerable Web Application which is then sent either to the victim or to the web server depending on which type of cross-site scripting the attacker is using. Now as the victim logs into this web app by giving the user-id and password, the server of this application will move the cookie of that specific session to the victim's browser. Then the victim browses the malicious JavaScript Code, this malicious script is executed either when the victim visits that web page or when the victim tries to access a piece of the site, so anyhow it gets executed on its browser and thus the cookies of the victim’s session will be transferred to the attacker ‘s server. Now lastly, these cookies will be utilized by the attacker to get into the account of the victim.

There could be numerous courses through which XSS attack is possible some of which are URL XSS which has the malicious code installed in the URL and gets executed when the client login, or input fields which permits to type information which will stay on the site. To encode this code, an attacker utilizes various methods so it seems legitimate to the client, while it isn't. For example, posting promotions, ads or pop ups through which the hacker could encode the code to look authentic to the client.

Example of a Malicious JavaScript Code for Stealing the Cookies:

```
<script type="text/javascript">
document.location='http://<ip>/cookiestealer.php?c='+document.cookie;
 </script>
```

4. Architecture for Detection and Prevention of XSS Attack

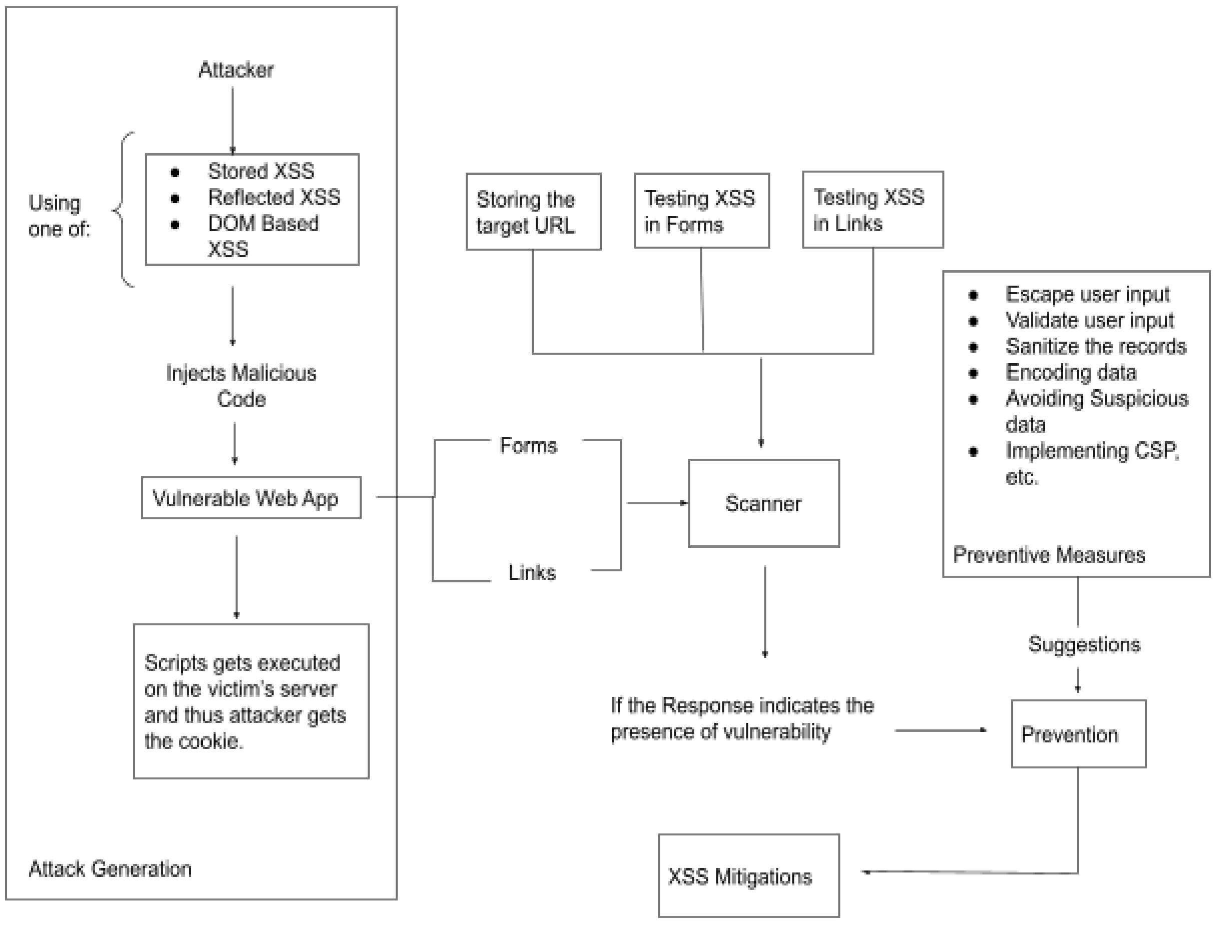

Figure 1. Architecture Diagram of XSS Attack, Detection and Prevention

<description needed><proposed xss attack and detection and called figure 1>

4.1. Detection

Now, there are a number of vulnerabilities and there are a number of ways to discover and exploit them. But the general idea of this vulnerability scanner which can be used to discover a large number of vulnerabilities is the same.

Basic Architecture of our Vulnerability Scanner:

1- Go into every possible page of the site.

2- Look for ways to send data to the web application.

Two main methods to send data:

(a) Through the URL (Usually using GET)

(b) Through input elements (Usually using POST)

3- Send payloads to discover vulnerabilities.

4- Analyse the response to check if the website is vulnerable.

General steps would be the same regardless of the vulnerability. To verify the presence of XSS vulnerability two functions will be included, one of which will test the XSS in forms and other will test the XSS in links.

- Storing the target url in the variable target_url after importing the scanner code.
- Storing the links to be not included while crawling the site in the variable links_to_ignore which could include for example a logout page.
- Initializing a dictionary with username and password.
- Running the scanner code.
- Extracting forms and storing them in the variable forms.
- Printing all the forms.
- Running the two functions in the scanner code specially designed to test the xss in forms and links which goes by name - test_xss_in_form and test_xss_in_link.
- Printing the response, we got after running those functions which would notify us if the xss vulnerability is present or not.

Algorithm for the driver code

4.2. Prevention

To defend most of the attacks from XSS vulnerabilities, we can follow these practices:

Escape user input - Escaping method to convert the key characters within the information that a page receives, to prevent the information from being interpreted in any malicious manner. Not allowing the special characters to be rendered. For example, there are special characters like greater than (>) symbol, smaller than (<) symbol which are generally used in tags or in malicious script or may be the percentage symbol (%) so the first thing we can do is escaping these characters by taking off the special feature of these character and make them just another text character.

Validate user input - Treating every information that originates from outside the system as untrusted as the attacker has complete control on what input arrives, so we need to assume that every input is a threat. Validating all the entered information is important.

Validation is necessary especially where we have a field of login where we can for example enter username and password, also we need to use data validation in case of email Ids, because in the generic format of an email ID there must be a username, an '@' symbol, so we can use data validation to avoid cross-site scripting attacks.

Sanitizing records - Looking for the undesirable data, along with HTML tags which can be unsafe. Keeping only the safe facts and eliminating other malicious things from the records. Some of the web pages are used to sanitize data by eliminating any script tag found and also by using regular expressions to eliminate all the script tags that can be generated.

Encoding the output - as in when we give the script tag and alert as the input in the malicious script, the arrow symbols for example will be treated as the arrow symbols here instead we can URL encode them so the arrow symbol will be something like %25 so that it's no longer a malicious script.

Response headers - we must use the right response headers, we must decide what the response header should be, we must decide what data can be sent or what data can be received through the response headers.

Finally, we could use Content Security Policies (also known as CSP standards) to avoid cross-site scripting.

These prevention methods although cover most XSS attack vectors, but they don't cover the entirety.

4.3. Environment Setup

We used DVWA which is Damn Vulnerable Web App to test our Scanner on. Cyber security professionals, Ethical hackers test/run their codes on this tool. It is a great platform for us to learn about web app security in a safe way. We need not take any permission from others and is totally legal, and overall, it is suited for both beginners and advanced users.

We used Beautiful Soup (Python Library) for getting the metadata from the sites and using them in the scanner. BeautifulSoup is used to extract data from HTML and XML files including URLs, texts, forms. In Linux, we can install the Beautiful Soup library of python using the system package manager; just the right version of pip must be made sure of. It transforms a complex HTML document into a tree of Python objects.

5. Implementation and Result

5.1. Implementation Details of Our Scanner:

We used Beautiful Soup - Python library for pulling data out of HTML and XML files. It works with the parser to provide ways of navigating, searching, and modifying the parse tree.

Our scanner code includes functions to extract links, extract forms, crawling function to find all the possible pages on the site, a run scanner function which is a generic method that is used to discover any vulnerability and two functions to test the XSS in the forms and links.

Basically, what we are doing is we are creating a new variable called is_vulnerable_to_xss and we are setting the value of this variable to whatever value returned from those two methods that tests XSS in the form and the links. Now this method’s response will indicate that the passed form or url contains a vulnerability or not.

All we have to do is just mention the target Website and it'll automatically map all the links that exist in this website, then it'll automatically extract all the forms and links, and then it'll start submitting payloads to discover vulnerabilities and then will show us if the vulnerability exists or not.

Logic	**Output Low**	**Output Medium**	**Output High**
def test_xss_in_link(self, url): xss_test_script = "<sCript>alert('test')</scriPt>" url = url.replace("=", "=" + xss_test_script) response = self.session.get(url) html = response.content.decode('ISO-8859-1') return xss_test_script in html	[<form action="#" method="GET" name="XSS"> <p>What's your name?</p> <input name="name" type="text"/> <input type="submit" value="Submit" /> </form>] True True	[<form action="#" method="GET" name="XSS"> <p>What's your name?</p> <input name="name" type="text"/> <input type="submit" value="Submit"/> </form>] True True	[<form action="#" method="GET" name="XSS"> <p>What's your name?</p> <input name="name" type="text"/> <input type="submit" value="Submit"/> </form>] False False

def test_xss_in_link(self, url):
 	
xss_test_script = "<script>alert('test')</script>"
 	
url = url.replace("=", "=" + xss_test_script)
 	
response = self.session.get(url)
 	
html = response.content.decode('ISO-8859-1')
 	
return xss_test_script in html
	[<form action="#" method="GET" name="XSS">
	
	<p>What's your name?</p>
	
	<input name="name" type="text"/>
	
	<input type="submit" value="Submit" />
	
	</form>]
	
	True
	
	True
		[<form action="#" method="GET" name="XSS">
		
		<p>What's your name?</p>
		
		<input name="name" type="text"/>
		
		<input type="submit" value="Submit"/>
		
		</form>]
		
		False
		
		False
			[<form action="#" method="GET" name="XSS">
			
			<p>What's your name?</p>
			
			<input name="name" type="text"/>
			
			<input type="submit" value="Submit"/>
			
			</form>]
			
			False
			
			False

Table 1. This table defines the logic on which the Scanner is been created

<scanner output and description, detection of xss >

5.2. Measures to Prevent XSS Attack

Step	**Description**
1	Escaping user input

2	Validating user input
3	Avoid inserting suspicious data
4	Encoding (HTML elements and attributes, JS Data, CSS Data)
5	Encoding URL Values
6	Sanitizing Records
7	Implementing Content Security Policy (CSP)

Table 2. Steps to Safeguard against XSS attack

S. No	**Name of the Data**
1	The URL
2	HTTP referrer objects
3	GET parameters from a form
4	POST parameters from a form
5	Window.location
6	Document.referrer
7	Document.location
8	Document.URL
9	Document.URLUnencoded
10	Cookie & Headers data

Table 3. List of data that must be Sanitized before using them to create a website

S. No	**Special Characters**	**Encoded Entity**
1	&	&amp
2	<	&lt
3	>	&gt
4	“	&quot
5	‘	&#x27
6	/	&#x2F

Table 4. Applying XSS Filter (Encoding)

5.3. Other suggestions

S. No	**Scanner Name**	**Description**
1	Arachni	Not only XSS, it can detect vulnerabilities including SQL injection, remote file inclusion, and several others.
2	X5S	Using this tool we need to manually find the injection point and then check where XSS might be in the app as it is not an automatic tool.
3	Wfuzz	It brute forces GET and POST parameters to test against injections.

4	WebScarab	It is a Java-based framework that analyses web apps using HTTP protocols.
5	W3af	It is a web app-based attack and audit framework and was developed using Python.
6	Wapiti	It tries to inject payloads and see if a script is vulnerable detecting multiple vulnerabilities.
7	Vega	Vega performs security testing of a web app using a GUI based environment.
8	Grabber	It is a web app scanner and detects many security vulnerabilities.

Table 5. List of Open Source Vulnerability Scanner

S. No	**Name**	**Description**
1	xss_clean.php filter	It cleans various URF encodings and exploits.
2	XSS HTML Filter	It can be used to sanitize user input and is a filter for Java.

3	PHP AntiXSS	It adds a layer of protection from cross-site scripting by detecting the encoding of the data which requires filtering.
4	HTML Purifier	It is a standard HTML filtering library, and it removes malicious codes.
5	xssprotect	It creates an HTML tag tree of a particular page and filters out the improper attributes.

Table 6. List of Open Source Libraries for Preventing from XSS Attacks

Further as to limit the dangers related with XSS, developers ought to encode all fields while showing them in the program. In most of the cases the source of XSS are external outdated libraries. As part of a defence in depth strategy, ensure that cookie properties and security headers are set accordingly. Also, regular tests would help identify such vulnerabilities and thus improve the security of the web applications.

6. CONCLUSION

With the internet expanding, the crimes have also taken a digital form. XSS attacks are one of the highly damaging ones with critical vulnerabilities. One of the main attack motives is to get the session id and use those cookies and sessions to view or use the websites. It is used to steal sensitive information and can be the initial step a hacker would be waiting for in order to access a computer. We should always be aware of vulnerable web apps. Functions intending to edit HTML contents could exploit the XSS attack.

Our assessment and results are based on the XSS attack methods and how they can be detected through the implemented code. Understanding the various types of XSS attacks, methodology and their damages shows us the direction we must proceed in, for a secure environment. The DVWA platform was used with various vulnerabilities in place and better prevention methods for better security variation. Based on the working of XSS attacks and the vulnerabilities, we described various preventive measures. Detecting the vulnerable part and fixing makes software secure. While coding we have also suggested certain guidelines to follow for the code to be inherently secure.

REFERENCES

[1] P. Chaudhary, B. B. Gupta and S. Yamaguchi, "XSS detection with automatic view isolation on online social network," 2016 IEEE 5th Global Conference on Consumer Electronics, Kyoto, 2016, pp. 1-5, doi: 10.1109/GCCE.2016.7800354.

[2] Bisht P, Venkatakrishnan VN. XSS-GUARD: precise dynamic prevention of cross-site scripting attacks. InInternational Conference on Detection of Intrusions and Malware, and Vulnerability Assessment 2008 Jul 10 (pp. 23-43). Springer, Berlin, Heidelberg.

[3] A. Stasinopoulos, C. Ntantogian and C. Xenakis, "Bypassing XSS Auditor: Taking advantage of badly written PHP code," 2014 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Noida, 2014, pp. 000290-000295, doi: 10.1109/ISSPIT.2014.7300602.

[4] A. Shrivastava, S. Choudhary and A. Kumar, "XSS vulnerability assessment and prevention in web application," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), Dehradun, 2016, pp. 850-853, doi: 10.1109/NGCT.2016.7877529.

[5] OWASPAntiSamyProject-OWASP:2016.
https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project.

[6] T. A. Taha and M. Karabatak, "A proposed approach for preventing cross-site scripting," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, 2018, pp. 1-4, doi: 10.1109/ISDFS.2018.8355356.

[7] O. Ismail, M. Eto, Y. Kadobayashi, and S. Yamaguchi. A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. In 8th International Conference on Advanced Information Networking and Applications (AINA04), March 2004

[8] K. Pranathi, S. Kranthi, A. Srisaila and P. Madhavilatha, "Attacks on Web Application Caused by Cross Site Scripting," 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, 2018, pp. 1754-1759, doi: 10.1109/ICECA.2018.8474765.

[9] A. Shrivastava, V. K. Verma and V. G. Shankar, "XTrap: Trapping client and server side XSS vulnerability," 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, 2016, pp. 394-398, doi: 10.1109/PDGC.2016.7913227.

[10] XSS (Cross Site Scripting) Prevention Cheat Sheet – OWASP: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

[11] M. Dayal Ambedkar, N. S. Ambedkar and R. S. Raw, "A comprehensive inspection of cross site scripting attack," 2016 International Conference on Computing, Communication and Automation (ICCCA), Noida, 2016, pp. 497-502, doi: 10.1109/CCAA.2016.7813770.

[12] Graff, M. and Van Wyk, K.R. 2003. Secure coding: principles and practices. O'Reilly Media, Inc.

[13] M. Singh, P. Singh and P. Kumar, "An Analytical Study on Cross-Site Scripting," 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 2020, pp. 1-6, doi: 10.1109/ICCSEA49143.2020.9132894.

[14] I. Dolnák, "Content Security Policy (CSP) as countermeasure to Cross Site Scripting (XSS) attacks," 2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, 2017, pp. 1-4, doi: 10.1109/ICETA.2017.8102476.