2016 2nd International Conference on Next Generation Computing Technologies (NGCT-2016)
Dehradun, India 14-16 October 2016

XSS Vulnerability Assessment and Prevention in Web Application

Ankit Shrivastava
Department of Computer Science and Engineering
Manipal University Jaipur
Rajasthan, India
Email:ankitshrivastavaieee@gmail.com

Santosh Choudhary
Department of Computer Science and Engineering
Manipal University Jaipur
Rajasthan, India
Email:shellysamota09@gmail.com

Ashish Kumar
Department of Computer Science and Engineering
Manipal University Jaipur
Rajasthan, India
Email:aishshub@gmail.com

***Abstract*—Cross site scripting (XSS) is a type of scripting attack on web pages and account as one of the unsafe vulnerability existed in web applications. Once the vulnerability is oppressed, an intruder advances intended access of the authenticate user's web-browser and may perform session-hijacking, cookie-stealing, malicious redirection and malware-spreading. As prevention against such attacks, it is essential to implement security measures that certainly block the third party intrusion. Recently the most dangerous attacks are reflected and DOM based cross-site scripting attacks because in both cases attacker attack using server side scripting and do forgery over the network, it is hard to detect and therefore it must be prevented. Vulnerabilities of websites are exploited over the network through web request using GET and POST method. In this paper, we are focusing on injection, detection, and prevention of stored based XSS reflected XSS and DOM based XSS.**

Keywords - XSS, Scripting, Injection Attack, Vulnerability, Web Applications, DOM based Cross Site

I. Introduction

Cross Site Scripting is a security bug that can affect web applications. This bug allows an attacker to inject their own malicious code into HTML pages that are displayed to the users. On successful execution of the malicious code, the system or website action or behavior can be completely changed. It also can steal user's private data or can be performed on behalf of the user and one of the utmost communal application-layer web attacks, it targets scripts embedded in a page, executed on the client-side rather than on the server-side [1]. XSS is a threat that occurs because of security flaws of client-side scripting languages like JavaScript and HTML. The model of XSS is to handle client-side scripts of a web-app to execute in the order preferred by the malicious manipulator. These kinds of manipulations can embed a script in a page that could be executed each and every time when it is loaded, or whenever an associated event is executed.

If a web page accepts data from users and includes it dynamically in web pages without any validation then XSS may arise. If an attacker can do successful attack then he can gain access to victim's account. Although the attacker can exploit the victim using the malicious script and bypass the security within the users session only. Another method to inject XSS is by sending a malicious link with an attractive level. The link doesn't seem harmful because of the encoding of the malicious portion of code. Generally, attackers send it using email, web message board and wait for clicking on that by victim [2].

There are three types of Cross-site Scripting attacks: persistent or stored, non-persistent or reflected and DOM-based. In DOM based and Non-persistent attack the attackers require a user who visits their malicious web page or click on a malicious link. The HTTP request posts automatically without the knowledge of the victim if the application sends the request using POST method only. In this case, vulnerable code will be submitted easily through request [3]. Upon submitting the malicious form or clicking on the malicious link, the XSS code will get executed again and it will get displayed on user's browser form. In Persistent attacks attacker injects when malicious code using the web form and it is stored in the database. Example message board, review form, webmail messages, web chat software and any user input form. The unsuspicious user is not necessarily need to interact with any additional link, he can simply view the web page containing code.

II. Literature Review

There are several studies and research has been done in last few years to find and prevent cross-site scripting related issues in the web application or network requests. Researchers have produced multiple solutions but XSS vulnerability still exists in the web application. Web applications still facing many XSS attacks and the most important one is session hijacking, attackers steal the victim's cookies details and they can use the same session.

A technique is used which include security throughout the application life cycle. The application lifecycle phases like design, development, deployment and runtime execution should be followed the OWSAP security guideline in a precise manner. Sometimes developer lack of knowledge can be the cause of security failure [4].

The ethics of vulnerability Research explored many security issues that are compromising with user's credential protection over the network. They suggested that the programmers need

978-1-5090-3257-0/16/$31.00 ©2016 IEEE

to understand the initial security so that we can remove it at first level. And also focuses on risk analysis of each and every scenario at initial level [5].

Tools and mechanism are invented to insure the web application security; it defines the policy that helps developers for developing a secure application. It says cross-site scripting is a result of improper filtering of user input and suggested careful removal of unwanted scripts or HTML tags that can be executed on the web application. And they also suggested content filtering on the server level and client level [6].

An automatic technique is also used to create input that exposed the XSS vulnerabilities. An automation tool ANDRILLA is introduced. It is based on input generation technique; it creates the malicious inputs and finds the security vulnerabilities including XSS [7].

The attackers attack on the server by manipulating the script, so content filtering is used that is using input filtering method. But in java script sometimes filter method fails to recognize the vulnerable input contents [8].

Sanitization method is also used to improve and prevent the user input. It consists of DOM, Input filed capture, input sanitizer, links, text area sanitizer and XSS notification. The system follows the layered approach to prevent the vulnerable contents [9].

III. Current Issues

The attackers use manipulation in java script to inject their malicious code to the server. The server cannot be able to differentiate between malicious or valid java script codes so that attackers easily send their invalid code to the server and server executes it [10].

Cookie Stealing: The attacker steals the user's cookie that contains the user's credentials and uses the victim's session using session id. There are several attack methods to steal the cookies if it is not secure [11].

Key logging: Using key logger the attacker can record and sent all keyboard activity of the user to their servers, like user id, password, card information and etc [12].

Phishing: The attacker can also steal the cookie using DOM based attack. He can steal the user information using the creation of fake login page by manipulation in DOM.

Although these all attacks are similar in some context, in all cases the attackers manipulate the existing java script code and inject the code. The request send to the web server is similar in both cases if it is the valid or invalid script, the server gets the request and serves the response as input script [13].

Entities of XSS attacks:

1) Website: Display the users requested HTML pages.
2) Where the website's user inputs stored.
3) Victim: Normal User
4) Attacker: The malicious user.
5) Attacker server: Attacker's web server.

IV. Proposed System

The proposed system is a study about cross-site scripting, there are several methods and approach to preventing the XSS attack but still, it is not prevented completely. So we use a hierarchical approach throughout the development process to secure the web application for both client side and server side because a single method and approach are not sufficient to handle it.

The proposed levels:

1) Initial level security testing is strictly suggested using both black box and white box testing technique.
2) Use of security testing tools to detect XSS, like Burp suite -is a powerful tool, using secure web application proxy we can detect and prevent the security vulnerabilities at first level.
3) For the client side, we should use the signature mechanism to recognize the malicious java script and verify the known attacks.
4) Use of application level firewall that handles the request between client and server.
5) While sending the request we can assign a unique token for each request that will compare between client and server, and it will be expired after completion of a request.
6) Never allow direct insertion of data into the script, HTML comment, attribute name, tag name, CSS.
7) Use escape method for HTML, attribute, java script, JSON, CSS, URL values, to prevent those characters that can be used to inject the script, like &, ', ", /, \, <, >, &lt, &gt, &amp, &quot, &#x27, &#x2F and etc, or we can also use auto-escaping template system.
8) Use of sanitization method to clean up HTML formatted text.
9) Use of HTTPOnly cookie flag.
10) We also suggest the use of spring MVC to develop java application. Spring is a powerful tool and manages many securities too.
11) Use of updated web browser.

V. Implementation and Explanation

Since the browsers do not allow accesses between sites (cross-site access). Therefore, intruders use different techniques to implement a cross- site attack.

A. STORED XSS

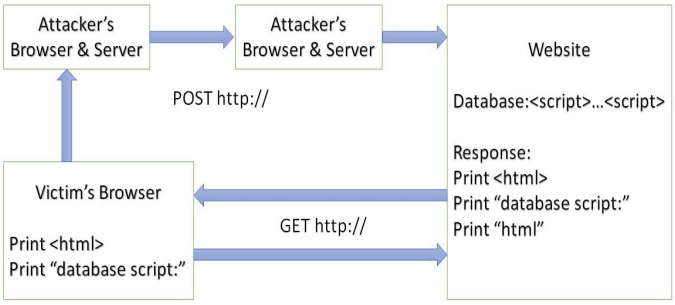

Fig. 1: Scenario diagram for stored XSS attacks.

Example: We inject malicious script in our application to steal the cookie and this could be performed by passing following script:

<script> window.location = 'http://google.co.in/?cookie='+ document.cookie </script>

This script will navigate the application browser to google.co.in. The URL includes the victim's browser cookie as a query parameter. Once the attacker gets the URL response with cookie he/she can use it to hack the victim's session.

Steps to steel cookie:

1) Here we are using a form of our application to inject the script into the database.

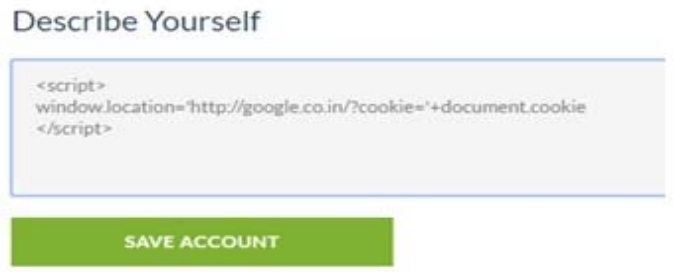

Fig. 2: User Input Form.

2) Next, we request a page from the website.
3) The website contains the malicious string from the DB in the response and sends it to the victim.
4) The victim's browser executes the malicious script inside the response, sending the victim's cookies to the attacker's server.

Fig. 3: URL redirection with session id.

B. REFLECTED XSS

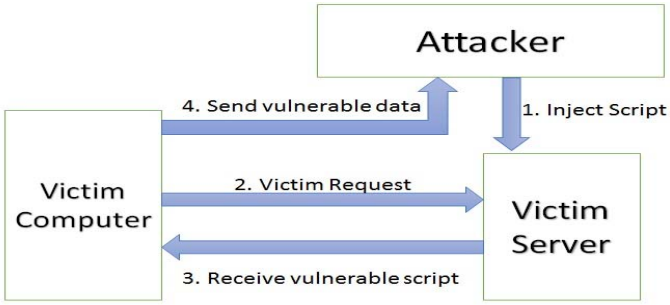

Fig. 4: Scenario diagram for reflected XSS attacks.

Steps for stealing cookies from test.mail.in using reflected XSS:

1) Sending a link to the victim:
   Button Text: Click Here To Get You Lucky Now
   http://mail.test.in/xss/Link_XSS.html
   Here, "mail.test.in" is victim website & "xss" is folder that contains JSP and HTML files.
2) When clicking on above link, the user will redirect to the given link http://mail.test.in/xss/Link_XSS.html, after the JSP file and the HTML file will be called from attacker server. This link will display cookie as a parameter.

Fig. 5: URL redirection with session id.

3) The attacker can view this session id on his server console.

Fig. 6: Attacker server receives the session id.

Other reflected Issues on test email server, checked using burp tool.
Example:http://10.11.12.13/2000/admin/ajax/IntermediateAddUser.jsp [SetComFonNam parameter]

TABLE I: Issue Details

Severity:	**High**
Confidence:	**Certain**
Host:	**http://10.11.13.111**
Path:	**/2013/admin/ajax/IntermediateAddUser.jsp**

The value of the SetComFonNam request parameter is copied into a JavaScript string which is encapsulated in single quotation marks. The payload 62370'%3balert(1)//821637fdb was submitted in the SetComFonNam parameter. This input was echoed as 62370';alert(1)//821637fdb in the application's response.

C. DOM BASED XSS

The following page http://test.mail.org.in contains the below code:

<script>
document.write("<b>Current URL<b>:"+document.baseURI);
</script>

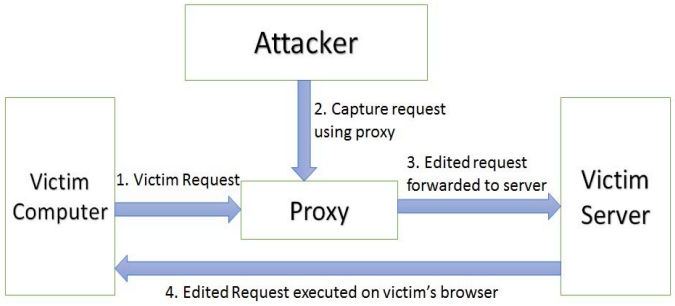

Fig. 7: Scenario diagram for DOM based XSS attacks.

http://test.mail.org.in/test.html#<script>alert(1)</script>, its simple enough JavaScript code that will get executed, If we look at the source of the page, we wont see <script>alert(1)</script> because its all happening in the DOM and done by the executed JavaScript code.

DOM based XSS using an example of test.mail.org.in:

1) Refresh web page and capture request in burp suite.
2) Capture request in burp suite.
3) Edit and add 84960';alert(1)//467ffe1de this as a parameter and forward the request.
4) Hence XSS had been performed.

Fig. 8: XSS alert on users web browser.

VI. Result Analysis

Our assessment and result are based on the study of previous research and their implementation in the real scenario. We tested many web applications and found high priority cross site scripting issues. Although we have many approaches available now to prevent XSS, but many web applications are still vulnerable.

In existing approach normally developers user java script validation or user input filter to prevent client side malicious inputs, for the output they use escape method and sanitation method. Some more secure web applications use application level firewall to filter user request. But still, we are not able to stop XSS attacks.

In our assessment, we found that it is not sufficient in the prevention of more dangerous XSS payloads. The use of one or two existing approaches can stop the direct malicious inputs from the web browser, but not strong enough to handle middleware attacks.

In our system, we are using java script validation for user input, java script signature mechanism to identify valid java script, assigning a unique token for client-server request during communication, using escape method to prevent script characters, sanitization method to clean-up HTML text.

And also, we suggest the use of strong application level firewall, use of HTTPOnly cookie flag, use of proper security testing and also the use of scanner tool to detect XSS payloads in all parameters, headers and path, use of strong SPRING tool to develop java application and use of updated web browser.

The proposed system help developer in the handling of XSS issues and it also help tester in detection. We suggested this hierarchical approach because a single level of security is not sufficient in the prevention of cross site scripting issues. Our system gives an effective result for prevention of XSS vulnerabilities.

VII. Conclusion

Cross-site scripting is one of the most hazardous website vulnerabilities. It is used to harm web applications and users. It harms users by sending or inserting malicious code into application database, mostly it is used to perform attacks like session hijacking. We also know that patching XSS is possible but we can never be completely sure that no one can break our filter. Attackers always find their ways to break application security. If we really want to make a hard-to-crack XSS filter, we have to analyze more on all XSS patterns and then we can use our prevention technique efficiently. After efficient analysis and using better prevention technique, we can stop or fix the dangerous xxx web application vulnerabilities. We should not depend on a single technique or approach, we should use a multilayer security and also we should have focused on initial level security. Use of SSL must be recommended to insure secure interaction between client and server.

References

[1] A. Singh and S. Sathappan, "A survey on xss web-attack and defense mechanisms," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 3, pp. 1160–1164, 2014.
[2] V. S. Patil, D. G. R. Bamnote, and S. S. Nair, "Cross site scripting: An overview," *IJCA Proceedings on International Symposium on Devices MEMS, Intelligent Systems and Communication*, no. 4, pp. 19–22, 2011.
[3] V. Nithya, S. L. Pandian, and C. Malarvizhi, "A survey on detection and prevention of cross-site scripting attack," *International Journal of Security and Its Applications*, vol. 9, no. 3, pp. 139–151, 2015.
[4] J. D. Meier, "Web application security frame," Oct. 19 2010, uS Patent 7,818,788.
[5] B. Schneier, *Bruce Schneier on Trust Set*. John Wiley & Sons, 2014.
[6] D. J. Scott, "Abstracting application-level security policy for ubiquitous computing," Ph.D. dissertation, University of Cambridge, 2005.
[7] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of sql injection and cross-site scripting attacks," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 199–209.
[8] N. Mishra, S. Chaturvedi, C. Dewangan, and S. Jain, "Solving false positive problem in client side xss filter," 2014.
[9] D. Patil and K. Patil, "Client-side automated sanitizer for cross-site scripting vulnerabilities," *International Journal of Computer Applications*, vol. 121, no. 20, 2015.
[10] S. Shalini and S. Usha, "Prevention of cross-site scripting attacks (xss) on web applications in the client side," *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 4, 2011.
[11] T. V. Kasture, P. P. Dixit, P. S. Ovhal, G. Sathe, and N. A. Zambre, "Multiple prevention techniques for different attacks in web application."
[12] T. S. Mehta and S. Jamwal, "Model to prevent websites from xss vulnerabilities," *IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 6, no. 2, pp. 1059–1067, 2015.
[13] S. K. Singh and R. Shrivastava, "Client side filter enhancement using web proxy," 2014.