ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

MITIGATING CROSS-SITE SCRIPTING ATTACKS WITH A CONTENT SECURITY POLICY

PRIYADHARSHINI.N [1] M.MOHANAPRIYA [2]

1. PG Student, Dept.of Computer Applications, VSB Engineering College, Karur.
2. HoD, Dept.of Computer Applications, VSB Engineering College, Karur.

Abstract

A content security policy (CSP) can help Web application developers and server administrators better control website content and avoid vulnerabilities to cross site scripting (XSS). In experiments with a prototype website, the authors' CSP implementation successfully mitigated all XSS attack types in four popular browsers. Among the many attacks on Web applications, cross site scripting (XSS) is one of the most common. An XSS attack involves injecting malicious script into a trusted website that executes on a visitor's browser without the visitor's knowledge and thereby enables the attacker to access sensitive user data, such as session tokens and cookies stored on the browser.1 With this data, attackers can execute several malicious acts, including identity theft, key logging, phishing, user impersonation, and webcam activation. **Content Security Policy** (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP is designed to be fully backward compatible; browsers that don't support it still work with servers that implement it, and vice-versa. Browsers that don't support CSP simply ignore it, functioning as usual, defaulting to the standard same-origin policy for web content.

1.INTRODUCTION

A primary goal of CSP is to mitigate and report XSS attacks. XSS attacks exploit the browser's trust of the content received from the server. Malicious scripts are executed by the victim's browser because the browser trusts the source of the content, even when it's not coming from where it seems to be coming from. CSP makes it possible for server administrators to reduce or eliminate the vectors by which XSS can occur by specifying the domains that the browser should consider to be valid sources of executable scripts. A CSP compatible browser will then only execute scripts loaded in source files received from those whitelisted domains, ignoring all other script (including inline scripts and event-handling HTML attributes). As an ultimate form of protection, sites that want to never allow scripts to be executed can opt to globally disallow script execution. Even major application services such as Facebook, Google, PayPal, and Twitter suffer from XSS attacks, which have grown alarmingly since they were first reported in a 2003 Computer Emergency Response Team advisory. The Open Web Application Security Project ranked XSS third on its 2013 list of top 10 Web vulnerabilities, calling it the “most prevalent Web application security flaw. Underscoring the widespread risk of XSS intrusions, WhiteHat Security’s May 2013 Web Security Statistics Report noted that 43 percent of Web applications were vulnerable to this kind of attack. Researchers have proposed a range of mechanisms to prevent XSS attacks, with content sanitizers dominating those approaches. Although sanitizing eliminates potentially harmful content from untrusted input, each Web application must manually implement it—a process prone to error. To avoid this problem, we use a different technique. Instead of sanitizing harmful scripts before they are injected into a website, we block them from loading and executing with a variation of the content security policy (CSP), which provides server administrators with a white list of accepted and approved resources. The Web application or website will block any input not on that list and thus there is no need for sanitizing. The white list also guards against data exfiltration and extrusion—the unauthorized downloading of data from a website visitor’s computer.

2. LITERATURE SURVEY

“Noxes : A Client-Side Solution for Mitigating Cross-site Scripting Attacks ” by Engin Kirda,Christopher Kruegel,Giovanni Vigna and Nenad Jovanovic

We highlight the relevant related works on XSS attack detection which is either at client or the server side. The literature summary provides different detection approaches either for persistent or non-persistent cross-site scripting attacks. The authors have proposed first client side solution for mitigating against XSS attacks. The Noxes tool acts as a web proxy and utilizes both manual and automatically generated rules. The user is allowed to create the filter rules for web requests. The rules can be created in three ways: manual creation, firewall prompts and snapshot mode. The disadvantage of this tool is it suffers from low reliability and the inclusion of benign HTML is prohibited. It requires user intervention to accept or deny requests.

Prithvi Bisht and V.N.Venatakrishnan provides aprevention mechanism for XSS attacks on the server side. The shadow pages are generated for every HTTP response in this approach. The purpose of shadow pages is to obtain intended set of authorized scripts that match up with HTTP response. The disadvantage of this approach is it attempts to sanitize unsafe output but influences the web browser parsers to infer unsafe HTML data. It is vulnerable to threats that utilizes browser parse quirks.

Peter Wurzinger,Christian Platzer,Christian Ludl , Engin Kirda and Christopher Kruegel has proposed a server-side mechanism for detection and prevention against XSS attacks. This technique comprises of a reverse proxy which will intercept all the HTML responses. It overcomes the disadvantage of XSS-Guard by detecting malicious embedded javascript code. It can detect the aberrant between benign and injected javascript code. This approach is not suitable for high performance web-service and it is limited to detect maliciously injected content to javascript.

Hossain Shahriar and Mohammad Zulkernine has proposed a server side framework for detection of XSS attack based on boundary injection and policy generation. It suffers from zero false negative.This technique is efficient and does not require any modification of server and client side entities but response delay increases due to increment of policy checking.

Imran Yusof and Al-Sakib Khan Pathan has proposed a client side solution by applying the pattern filtering approach to prevent persistent XSS attack. This approach is effective but these rules does not work for non-persistent XSS attack. If the filtering module fails o detect any malicious script then it will stored and executed in the database.

Shashank Gupta, B.B Gupta proposed a server side approach for detection and mitigation of XSS attacks in javascript code.It is based on injecting features, generating rules and allows insertion of sanitization routines for the discovery of XSS attacks. The drawback of XSS-Safe is it detects the relationship between stored and injected features in the source code of Javascript.

Imran Yusof and Al-Sakib-Pathan proposed a technique that blocks malicious scripts from loading and executing with a variation of the content security policy (CSP), which provides server administrators with a white list of accepted and approved resources. This approach protects the website visitor from unauthorized downloading of the sensitive data stored in their browsers. The drawback of this approach is it will work only if the browser supports CSP and primary defense must involve validating the user inputs and encode user outputs.

2.1 EXISTING SYSTEM

Thus system will send request with identity. After that all the collected information will be send to collection database server. It not only protects clients from XSS attacks but also inform the vulnerable web servers. This mechanism requires minimal effort and low performance overhead. Also, it will prevent all the types of XSS attacks. The pattern filtering approach is only to prevent persistent XSS attack. This approach is effective but these rules does not work for non-persistent XSS attack. If the filtering module fails o detect any malicious script then it will stored and executed in the database.

2.1.1 Disadvantages

- How to use the collected information in database is not addressed.
- How to make system deployed universally has also not been addressed.
- It requires modifications in the frameworks or installation of additional frameworks.
- Approved scripts have to be identified by the website.
- There is no single policy for all the documents.
- Creating policies manually is a very tough task.
- This approach incurs runtime overhead due to interception of HTTP traffic.
- It requires user-defined security policies which can be labour-intensive.

2.2. PROPOSED SYSTEM

The existing approaches mostly focus on detection XSS attack either at client side or at the server side. So there is a need to come up with a solution that has the ability to detect Persistent as well as Non-Persistent XSS attack which will work both at the client and server side. This system has proposed a detection model that will validate the user provided inputs at the client side and the response pages from the server is also validated and then given back to the client. The proposed system is a detection model for XSS attack consisting both Persistent and Non-Persistent cross-site scripting attack. The proposed model has different architecture for client and server side.

2.2.1 Advantage of Proposed System

- ❖ Accuracy.
- ❖ Computational Efficiency.
- ❖ Scalability and Reliability.
- ❖ Web applications are utilized for security-critical services so they have turned out to be a well-liked and precious target for web-related vulnerabilities.XSS attacks allows the attacker to execute malicious script on the victim's browser thereby stealing user's sensitive information.
- ❖ Proposed approach is modelled in such a way that it validates the input at the client side. This technique works for both Persistent and Non-Persistent XSS attack. The server side approach provides validated output.

3. SYSTEM MODEL AND PROBLEM DEFINITION

Cross-site scripting is a type of computer security vulnerability that is found in web-based

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

applications which allows code injection by malicious web users into any webpage that is viewed by other users. The term "Cross-site scripting", originated when a malicious website could potentially load a website onto another window and then use JavaScript to read or write information on the other website, which was later redefined as injection. XSS is made possible due to the fact that faulty coding causes XSS holes (vulnerabilities on websites that allows attackers to avoid security measures) in the client-side script that allows for insertion of malignant code.

During an attack, "everything looks fine" to the end user, but in actuality they are subject to a wide variety of threats. These vulnerabilities are exploited by attackers to by pass access controls such as the same origin policy. XSS is a potentially dangerous vulnerability that is easy to execute and very long and arduous to repair. The most frequent kinds of web applications that are victimized by XSS attacks are search engines, discussion boards, web-based emails, and posts. Even the most well-known websites in today's world like Google, Yahoo!, MySpace, Facebook, PayPal, and WikiPedia were once victims and still are very susceptible to many kinds of XSS attacks.

To avoid the web application attacks, the web browser security model is built on the same origin policy that isolates one origin from the other thus providing the developers a safe sandbox environment to build these applications in which the code from one origin (http://self.com) has access to only http://self.com data and the code from other origin (http://other.com) is not permitted to access http://self.com data. But the attackers by-pass this policy by exploiting cross site scripting vulnerabilities in the web applications. He injects his own script into the web applications and later this injected script will get embedded along with the actual intended response from the website whenever any user visits that particular web page.

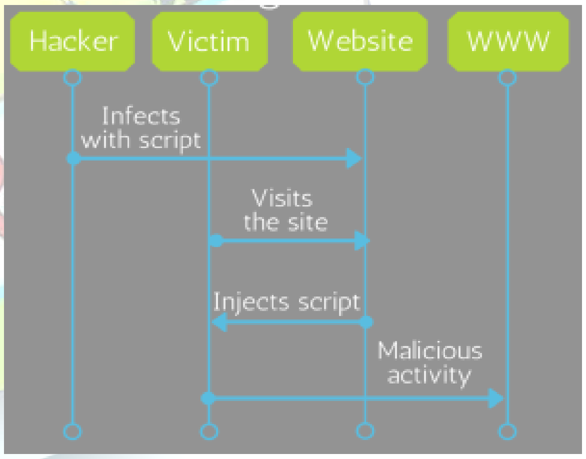

Fig.5.1.1 High Level View of Cross Site Scripting

The victim‟s browser executes all of the code that shows up on a page as being legitimately part of that page‟s security origin since the browser is not able to differentiate between the injected and the intended code. Thus, Cross-Site

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

Scripting attack (XSS) is a code injection attack performed to exploit the vulnerabilities existing in the web applications by injecting html tag / JavaScript functions into the web page so that it gets executed on the victim"s browser when one visits the web page and successfully accesses to any sensitive victim"s browser resource associated to the web application (e.g. cookies, session IDs, etc.). Successful cross site scripting can result in serious security violations for both the web site and the user.

3.1 OVERVIEW OF THE SYSTEM

Web applications are generally classified into two types; they are static web applications and dynamic web applications. Static web applications are those which does not interact with server (or database) and display the static content to the users. Dynamic web applications are those which interact with the server and satisfy the request of the client, for example, a sample login page which verifies the username and password of the user by interacting with the database in which the user credentials are stored .Cross site scripting attacks are the type of attacks which enables the attackers to steal the client side sensitive information like cookies etc.. These kind of attacks are generally done by injecting the client side vulnerable scripts into the areas which communicate with the servers or the databases like search fields, comment box etc.. By stealing user sensitive information attackers can bypass the access controls like same origin policy.

3.2 TYPES OF CROSS SITE SCRIPTING ATTACKS

There are mainly three types of cross site scripting attacks. They are:

i. **Non persistent Attacks:** It is the most common type of web vulnerability and is also termed as reflected XSS attack or type 1 XSS because the attack is carried out in a single request/response cycle. This attack is done mostly in HTTP query parameters given by the users and is used by scripts on the server side and display the results without sanitizing the query. These attacks are easy to identify and attacker initially checks whether a particular web application is vulnerable or not by performing these attacks. These attacks are not so devastating since these do not show impact on the server.

ii. **Persistent Attacks:** It is the more dangerous type of XSS attack and is commonly termed as stored XSS attack or type 2 XSS because the attack is carried out in two requests one for injecting the malicious code and store it in the web server and the other for the users(victims) to load the page which is malicious. In this attack, the attacker stores the malicious script on the server side permanently and when the users unknowingly or without proper knowledge make

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

the script active he/she will be a victim of the attack.

iii. **DOM based Attacks:** In these attacks, the vulnerability appears in the document object model. In type 1 and type 2 XSS, the dangerous payloads are in the response page but in this type of attack, the dangerous payload is not in the response page and the source code of the HTML page is similar to the response page. These attacks are done by the use of document.write() and other such similar functions.

3.3 SCRIPT FILTERING ALGORITHM

This algorithm works best because here the mechanism implemented deals with input given by the user. Whatever is the input given by the user is sanitized properly and displayed to the user.

```
Step 1: consider user input
Step 2: while(given user input)
If(user input contains any HTML specific tags)
Sanitize the input and store in the database
If(user input contains any special symbols)
Sanitize the input and store in the database
If(user input contains any script tags)
Sanitize the input and store in the database
If(user input contains any DOM objects)
Sanitize the input and store in the database
If(user input contains window objects or document objects)
Sanitize the input and store in the database.
If(user input contains any styling related code)
Sanitize the input and store it in the database.
Step 3: Take the user input and goto step 2
Step 4: Display the results.
```

For an attack to happen, the attacker tries to find the user input areas. The user input is given such priority because it is the only way for the user or client to interact with the server. So if the attacker can be successful in injecting the malicious code into the server an attack is guaranteed to happen. In order to prevent the attacker to have that privilege, sanitize the user input. If the user input contains any HTML specific tags like "<i>, <br>, <a> etc.. "sanitize the request and store it in the database. If the user input contains any special symbols which are generally used in script functions, they should be sanitized. If the user input contains any script tags which are one of the most serious ways of an attack to be possible, they should be properly sanitized. If the user input contains any styling related code then filter the code and store it in the database. Finally, it has to be restricted the redirection of a specific web application page to some other page through which it can stop most of the attacks. This can be done by

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

sanitizing the user input if it contains any window.location or document.refferer methods. If the above methods are not followed, the attacker tries to steal the valuable information of the users like cookies. Usually, if it considers any login page example sessions will be created for every user. The flaw of any browser is that it stores the session id in the form of a cookie. So, if the attacker steals this cookie he can enter into the web application as an authorized user and the results can be more devastating. [5] proposed a secure hash message authentication code. A secure hash message authentication code to avoid certificate revocation list checking is proposed for vehicular ad hoc networks (VANETs). The group signature scheme is widely used in VANETs for secure communication, the existing systems based on group signature scheme provides verification delay in certificate revocation list checking. In order to overcome this delay this paper uses a Hash message authentication code (HMAC).

3.4 MODULE DESCRIPTION

MODULE 1: Detection of Persistent XSS Attack

The Persistent XSS attack consists of five modules: Input Sanitizer, Filtering Module, Filtered Output, Attack Rule Library and Attack Repository.

1. Input Sanitizer

This block will check the incoming request and determine whether it contains any malicious scripts. The input sanitizer will check the presence of malicious code, if no then request is allowed else it is passed on to the Attack Rule Library.

2. Filtering Module

The filtering module receives input from the Input Sanitizer. The filtering module is capable to filter malicious scripts present in the Event Handlers, Data URI, Insecure Keywords, Character Escaping, Common words in XSS Payload and filtering XSS Buddies.

3.Filtered Output

After the filtering module completes its processing the resultant output will generated by this module.

4. Attack Repository

This block is responsible for storage of log records. After the input is sanitized, it will be passed to the filtering module and finally filtered output will be generated. The Attack repository will maintain the logs of all the filtered outputs for future use.

The main work of this block is to store the rules. Some examples of rules stored in the library will be as follows :

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

• <script>...DO NOT PUT UNTRUSTED DATA

..</script> directly in a script.

• <!_...DO NOT PUT UNTRUSTED DATA ..._>

inside an HTML comment.

• <div ...DO NOT PUT UNTRUSTED DATA ...=test

/> in an attribute name.

• Perform escaping of the URL before inserting untrusted or malicious data into HTML url parameter values.

MODULE 2 : Detection of Non-Persistent XSS Attack

The detection of Non-Persistent XSS attack consists of five blocks : Input Checker, Prevention using CSP(Content Security Policy), Notify Client ,Attack Rule Library and Attack Repository.

1. Input Checker

The input checker accepts the incoming request and checks for any malicious scripts from the Attack Rule Library. Suppose the incoming URL has malicious contents written inside script tag then the Attack Rule Library has a rule where the contents inside a script tag needs to be validated.

2. Content Security Policy

CSP is used to restrict the browser viewing your page so that it can only utilize resources downloaded from trusted sources. It has a white list of trusted source and browser contents. The main aim of CSP is to check whether to permit the browser from loading the website or not from its list of white sources.

3. Notify Client

This module will send an alert message to the client indicating that the website is not trusted and so the browser blocks it from execution. The rejected URL's will be stored in the attack repository for future use.

4. Attack Repository

This block is responsible for storage of log records. After the input is checked for presence of malicious scripts or tags ,it will be passed to CSP module and finally the client is notified with alert message in case of malicious website. The Attack repository will maintain the logs of all the blocked URL's for future use. The functioning of Attack Rule Library is similar to one mentioned in the detection of Persistent XSS attack.

MODULE 3 : Detection of XSS Attack at Server Side

The detection of Cross-site scripting attack at the server side has five modules: Feature

Injection,Policy Storage ,Web Server, Output Response Deviator, Sanitization and Feature Removal.

1.Feature Injection

This module is responsible for inserting an HTML or Javascript comment or features that does not modify the intended HTTP response or behaviours. The evaluation of features is to discover the presence of malicious injected contents.

2. Policy Storage

This module is responsible for storage of policies which represent the expected features such as number of tags, attributes, method names and arguments .The policies are given to the Output Response Deviator to check any variation between the actual and expected features.

3. Web Server

The web server represents the instrumented code with injected features that can be accessed from browsers. The request received by web server provides an initial response which is forwarded to the Output Response Deviator.

5. Output Response Deviator

6. This module is responsible for analyzing the initial response pages produced by web server .It checks whether any deviation is found between the actual and the expected features. XSS attack is detected if any deviation is found between the actual and expected features.

7. Sanitization

8. The main of sanitization is to remove the harmful scripts or contents. This module will remove the malicious contents and then give response to the client.

9. Feature Removal

10. In this step if no attack is detected by the Output Response Deviator then the boundaries or features could be removed and HTTP response is given to the client.

4. CONCLUSION AND FUTURE SCOPE

Web applications are utilized for security-critical services so they have turned out to be a well-liked and precious target for web-related vulnerabilities.XSS attacks allows the attacker to execute malicious script on the victim's browser thereby stealing user's sensitive information. The existing approaches mostly focus on detection XSS attack either at client side or at a server side. So there is a need to come up with a solution that can detect Persistent and Non-Persistent XSS Attack which will work both at the client and the server side. Thus our proposed approach is modelled in such a way that it validates the input at the client side. This technique works for both Persistent and Non-

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 17, March 2017

Persistent XSS attack. The server side approach provides validated output.

FUTURE SCOPE

In this work, it has been tried to restrict the XSS attacks with the help of code filtering algorithm. This algorithm works fine because it allows no script to store in the database and thus no script can be made executed. But, it mades the efforts to reduce the XSS attacks by means of cookie stealing which is not the only way of performing XSS attacks. In future, the same algorithm will be implemented to restrict attacks done through key logging etc

.REFERENCES

[1] Okin, Jonathan Robert. The information revolution: the not-for-dummies guide to the history, technology, and use of the World Wide Web. Ironbound Pr, 2005.

[2] Barth, Adam. "The web origin concept." (2011).

[3] http://www.acunetix.com/blog/articles/ non-persistent-xss as accessed on 11 March 2017.

[4] Jayamsakthi Shanmugam, Dr M. "Cross Site Scripting-Latest developments and solutions: A survey." Int. J. Open Problems Compt. Math 1.2 (2008).

[5] Christo Ananth, M.Danya Priyadharshini, "A Secure Hash Message Authentication Code to avoid Certificate Revocation list Checking in Vehicular Adhoc networks", International Journal of Applied Engineering Research (IJAER), Volume 10, Special Issue 2, 2015,(1250-1254)

[6]http://www.acunetix.com/blog/articles/dom-xss-explained as accessed on 11 March 2017.

[7] Matsuda, Takeshi, Daiki Koizumi, and Michio Sonoda. "Cross site scripting attacks detection algorithm based on the appearance position of characters." *Communications, Computers and Applications (MIC-CCA), 2012 Mosharaka International Conference on*. IEEE, 2012.

[8] Ruse, Michelle E., and Samik Basu. "Detecting cross-site scripting vulnerability using concolic testing." *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*. IEEE, 2013.

[9] Dong, Guowei, et al. "Detecting cross site scripting vulnerabilities introduced by HTML5." *Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on*. IEEE, 2014.

[10] Gupta, Shashank, and B. B. Gupta. "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art." *International Journal of System Assurance Engineering and Management* (2015): 1-19.