DOM BASED REFLECTED XSS ATTACK USING SOP

Manorama Singh and Abhishek Singh Chauhan

1M-Tech Student, 2Assistance Professor, Computer Science & Engineering, NRI Institute of Information Science & Technology (NIIST), Madhya Pradesh, India.

ABSTRACT:

Cross site scripting (XSS) attacks are mainly used securing web applications. XSS attacks categorized into three ways are: Stored XSS attack and Reflected XSS attack, DOM based XSS attack. In Stored XSS attacks is also called Type – I XSS or Persistent. Here, the malicious scripts permanently stored in Servers. Reflected XSS attacks known as Type – II XSS or Non-persistent. Here the attacker attacks the server indirectly through some web users. A cross-site scripting vulnerability used by attackers to bypass access controls such as the same-origin policy (SOP). Cross site scripting attacks are easy to execute but difficult to detect and prevent the data. In existing system XSS attack include some regular expressions to detect the malicious contents. In proposed system we developed DOM based filtering mechanism. In a DOM-based filtering XSS attack, the malicious data does not touch the web server.

Keywords: Cross-site Scripting, DOM based filtering, web Security

[1] INTRODUCTION

Cross-site scripting (XSS) is a type of web security vulnerability typically found in Web applications that accepts user inputs. XSS enables attackers to insert client-side script into Web pages viewed by other users. Restrictive access control mechanism like Same Origin Policy (SOP) can be bypassed using XSS attack. Reflected XSS attack is web security vulnerability where the attacker injects malicious client-side script into a web page. When a user visits a web page, the script code is downloaded and transparently run by the web browser. The malicious script inherits the user’s rights, authentication, and so on. XSS represents the majority of web-based security vulnerabilities. One reason for the popularity of XSS vulnerabilities is that developers of web-based applications often have little or no security background.

XSS protection can be configured for multiple types of request and response data – URL query parameters – URL encoded input (“POST data”) – HTTP headers – Cookies. The possibilities to manipulate HTML documents displayed by the browser with JavaScript or to influence the operation of the browser itself are dangerous features if misused. The misuse potential directly relates to the functions available for a malicious programmer. Unfortunately JavaScript provides full access to HTML documents using the document object model (DOM). A script therefore can modify at least the
DOM BASED REFLECTED XSS ATTACK USING SOP

document it is residing in arbitrarily: it is also possible to completely delete the document and create a totally different document. From an attackers point of view two things are of special interest: cookies associated to a document and access credentials. JavaScript also provides access possibilities to this information. The cookies associated to a document can be accessed using the function call document.cookie and application level access credentials are often acquired using form based login.

[2] TYPES OF CROSS-SITE SCRIPTING

To prevent the script code contained in a document loaded from some Web site accesses documents loaded from some other Web site, browsers do not allow access between documents loaded from different sites (i.e. cross-site access). Therefore attackers use other techniques to implement a cross site attack. In general there are currently three major categories of cross-site scripting. Others may be discovered in the future, however, so don't think this sort of misuse of Web page vulnerability is necessarily limited to these three types. Reflected Cross-Site Scripting attacks, Stored Cross-Site Scripting attacks and DOM based Cross-Site Scripting attacks.

REFLECTED XSS:

Probably the most common type of cross-site scripting exploit is the reflected exploit. It targets vulnerabilities that occur in some Web sites when data submitted by the client is immediately processed by the server to generate results that are then sent back to the browser on the client system. An exploit is successful if it can send code to the server that is included in the Web page results sent back to the browser, and when those results are sent the code is not encoded using HTML special character encoding thus being interpreted by the browser rather than being displayed as inert visible text. The most common way to make use of this exploit probably involves a link using a malformed URL, such that a variable passed in a URL to be displayed on the page contains malicious code. Something as simple as another URL used by the server-side code to produce links on the page, or even a user's name to be included in the text page so that the user can be greeted by name, can become a vulnerability employed in a reflected cross-site scripting exploit.

STORED XSS:

Also known as HTML injection attacks, stored cross-site scripting exploits are those where some data sent to the server is stored (typically in a database) to be used in the creation of pages that will be served to other users later. This form of cross site scripting exploit can affect any visitor to your Web site, if your site is subject to a stored cross-site scripting vulnerability. The classic example of this sort of vulnerability is content management software such as forums and bulletin boards where users are allowed to use raw HTML and XHTML to format their posts. As with preventing reflected exploits, the key to securing your site against stored exploits is ensuring that all submitted data is translated to display entities before display so that it will not be interpreted by the browser as code.

[3] LITERATURE REVIEW

“AntiSamy – A Java Solution for XSS Attacks” The increasing popularity of web applications has increased the threats and security vulnerabilities. One such vulnerability is due to XSS or Cross-Site
XSS is an attack technique that forces web application to echo the malicious executable code supplied by the attacker, which then loads in the results in the user’s web browser. The data supplied is usually the HTML/JavaScript code that executes at the client side.

“Large-scale detection of DOM-based XSS.” Cross-Site Scripting is an attack in which an attacker is able to inject his own JavaScript code into a Web application, in such a way that the code is executed within a victim’s browser in the context of the application.

“Server side API to secure XSS” Server side API for Cross-site scripting which differentiates XSS attack from simple script. Thus novice users can enjoy the safe and better experience of browsing without any surge of functionality, need of additional software or configuration at browser side. Developing such API also reduces burden to web administrators to safeguard their web applications from malignant XSS attacks.

“Same origin policy” The same-origin policy restricts how a document or script loaded from one origin can interact with a resource from another origin. It is a critical security mechanism for isolating potentially malicious documents. An origin is defined by the scheme, host, and port of a URL. Generally speaking, documents retrieved from distinct origins are isolated from each other.

“DOM based XSS Prevention Cheat Sheet” This paper refers to the HTML, HTML attribute, URL, and CSS Cheat sheet contexts as sub contexts because each of these contexts can be reached and set within a JavaScript execution context.

“Regular expressions considered harmful in client-side XSS filters” Cross-site scripting flaws have now surpassed buffer overflows as the world's most common publicly-reported security vulnerability. Browser vendors and researchers have tried to develop client-side filters to mitigate these attacks. We analyze the best existing filters and find them to be either unacceptably slow or easily circumvented. Worse, some of these filters could introduce vulnerabilities into sites that were previously bug-free. We propose a new filter design that achieves both high performance and high precision by blocking scripts after HTML parsing but before execution. Compared to previous approaches, our approach is faster, protects against more vulnerability, and is harder for attackers to abuse. We have contributed an implementation of our filter design to the Web Kit open source rendering engine, and the filter is now enabled by default in the Google Chrome browser.

“Dynamic prevention of cross-site scripting attacks” XSS attacks and analyze the reasons for the failure of filtering mechanisms in defending these attacks. We conclude that while filtering is useful as a first level of defense against XSS attacks, it is ineffective in preventing several instances of attack, especially when user input includes content-rich HTML.

[4] CONCLUSION

In our proposed system we plan to develop the PROBE mechanism. The PROBE mainly used to detect the malicious code and block the malicious users. In server side we are using DOM based XSS filtering mechanism. This XSS verifier filters the request and removes the malicious code, and then server is response to the user.
REFERENCES

[1] Sebastian Lekies, Ben Stock, Martin Johns "Large-scale detection of DOM-based XSS,” ACM Publication 978-1-4503-2477-9/13/11


