INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 3 ISSUE 1 (2017) PAGES 1648 - 1654
RECEIVED : 15.02.2017 PUBLISHED : 26.02.2017
February 26, 2017

A DUAL HYBRID APPROACH FOR DETECTING CODE INJECTION IN WEBPAGES WITH XSS ATTACKS

Divya. K .U[1] , Samundeeswari. M[2], Ananda Kumari. A[3]
[1] PG Student, Dept. of Computer science and Engineering, Priyadarshini Engineering College, Vaniyambadi,Tamilnadu, India,
[2]Associate Professor, Dept. of Computer science and Engineering, Priyadarshini Engineering College, Vaniyambadi, Tamilnadu, India,
[3]Associate Professor, Dept. of Computer science and Engineering, Priyadarshini Engineering College, Vaniyambadi, Tamilnadu, India.

Abstract:
Now a day's web application plays a vital role in internet such as internet banking, online shopping, etc. Cross site scripting (XSS) is one of the well known attack where malicious code is injected in trusted web application site and it is executed in web browser that access it. Credential data such as cookies information, password are steal from the user using this XSS.A numerous approach such as content security policy (CSP) has been proposed to avoid vulnerabilities of XSS. We propose a hybrid client server XSS detection and avoidance approach which involves server side filtration of malicious code injection as well as client side elimination of XSS code using standard policies in web site responses. This is a dual security measure were both web browsers as well as web server are involved in XSS detection and removal.

Keywords: Cross site scripting (XSS), content security policy (CSP)

1. INTRODUCTION

A cross-site scripting attack is an attack on web applications that tries to inject malicious codes to perform harmful things in trusted websites. A cross-site scripting attack will occurs when a web application executes a attacker script in the trusted web document. This is done when the user input not properly encoded and sanitized, this injected malicious code will be sent to the web browser and the browser has no way to know this. When the browser executes the code, a harmful things is performed on the web browser. XSS is used to hack cookies and session information of a valid user to perform session tracking [1].

Consider a example of search form code

```
<form action="look.php" method="get">
<input type="text" name="q" value="" />
<input type="submit" value="send" />
</form>
```

On the look.php the code will be

```
<h3>You looking for: <!--?php echo($_GET['q']) ?-->
```

Whatever a person look for, it will be displayed on the web page along with search results. if an attacker tries to inject malicious codefrom this side.

Search for

```
"><script>alert('XSS injection')</script>
```

If web application has nothing implemented to encode input and filter malicious scripts, it will take input as it is and then print on webpage where it will be called. So, at the keyword place, it will look like this:

```
<h3> You Searched for: "><script>alert('XSS injection')</script>
```

It will be executed by the browser and it will display an alert box saying "XSS injection."

Suppose there is a website with a messaging feature. In this website, users can send messages to their contacts. A basic form will look something like this:

```
<form action="sendmessageinfo.php" method="post"'>

<textarea name="message"> </textarea>

<input type="submit" value="send" />

</form>
```

When this form is submitted, the message will be stored in the database. Another person will see the message when he opens the message from the inbox. Suppose an attacker has sent some cookie-stealing script in the message. This script will be stored on the website as a message. When the other person tries to read the message, the cookie-stealing script will be executed and his session id is now on the attacker's side. With a valid session id, the attacker can hijack the other person's account.

2. LITERATURE SURVEY

This attack can be eliminated using the pattern filter in the server side and produces the proper sanitized output to the web browser thereby avoiding the XSS vulnerabilities.The detection and prevention involves analysis of various vulnerable worms in recent years such as Samy Worm, Space Flash Worm, Yamanner Worm, Twitter Worm, Facebook XSS worm, The Anthem Data Breach,etc and methods of preventing these kinds of XSS attacks.[3]In general, XSSattack is easily executed and difficult detect and prevent the attack. The reason for this is flexible HTML encoding schemes, Noxes,the first client-side solution to miti-gate cross-site scripting attacks. Noxes acts as a web proxy and uses both manual and automatically generated rules tomitigate possible cross-site scripting attempts. Noxes effec-tively protects against information leakage from the user's environment while requiring minimal user interaction and customization effort.[4]

Minimize trust on web browser content. A tool called BLUEPRINT that was working with several web applications. It has been evaluated against stress testand found that it is resistance to XSS attacks, excellent compatibility with web browsers and reasonable performance overheads.[5]A Browser-Enforced Embedded Policies (BEEP) in which a web site can embed a policy in its pages that says which scripts are allowed to run. The browser knows exactly when it will run a script, can enforce this policy perfectly. We have added BEEP support to several browsers, and built tools to simplify adding policies to web applications. We found that supporting BEEP in browsers requires only small and localized modifications, modifying web applications requires minimal effort, and en-forcing policies is generally lightweight.[6]A technique that uses X.509 certificates, and XACML for the expression of authorization policies. By this a developers and/or administrators of a given web application can specifically express its security requirements from the server side, and require the proper enforcement of such requirements on a compliant client. This strategy is seam-lessly integrated in generic web applications by relaying in the SSL and secure redirect calls.[7]

Types of Cross-site Scripting Attack

Cross site scripting is classified as.

(i) Non-Persistent Cross-site scripting attack

Non-persistent XSS is also known as reflected cross-site vulnerability. It is the most common type of XSS. In this, data injected by attacker is reflected in the response. If you take a look at the examples we have shown above, the first XSS example was a non-persistent attack. A typical non-persistent XSS contains a link with XSS vector.[7]

(ii) Persistent cross-site scripting attack

Persistent cross-site scripting is also known as stored cross-site scripting. It occurs when XSS vectors are stored in the website database and executed when a page is opened by the user. Every time the user opens the browser, the script executes. In the above examples, the second example of messaging a website was a persistent XSS attack. Persistent XSS is more harmful that non-persistent XSS, because the script will automatically execute whenever the user opens the page to see the content. Google's orkut was vulnerable to persistent XSS that ruined the reputation of the website.[7]

(iii) DOM-based cross-site scripting attack

DOM-based XSS is also sometimes called "type-0 XSS." It occurs when the XSS vector executes as a result of a DOM modification on a website in a user's browser. On the client side, the HTTP response does not change but the script executes in malicious manner. This is the most advanced and least-known type of XSS. Most of the time, this vulnerability exists because developers do not understand how it works.[7]

Proposed System

Client side based XSS solution

create a XSS filter in the web browser to filter the harmful malicious code from the web server .

There is a simple rule that you need to follow everywhere: Encode every datum that is given by a user. If data is not given by a user but supplied via the GET parameter, encode these data too. Even a POST form can contain XSS vectors. So, every time you are going to use a variable value on the website, try cleaning for XSS.

These are the main data that must be properly sanitized before being used on your website.

- The URL
- HTTP referrer objects
- GET parameters from a form
- POST parameters from a form
- Window.location
- Document.referrer
- document.location
- document.URL
- document.URLUnencoded
- cookie data
- headers data
- database data, if not properly validated on user input

First of all, encode all <, >, ‘ and “. This should be the first step of your XSS filter. See encoding below:

- & –> &
- < –> <
- –> >
- ” –> "
- ‘ –> '
- / –> /

For this, you can use the **htmlspecialchars()** function in PHP. It encodes all HTML tags and special characters.

```
$input = htmlspecialchars($input, ENT_QUOTES);
```

If the $input was= “><script>alert(1)</script>

This function would convert it into "><script>prompt(1)</script>

This line also helps when an encoded value is used somewhere by decoding it:

```
$input = str_replace(array('&amp;','&lt;','&gt;'), array('&amp;amp;','&amp;lt;','&amp;gt;'), $input);
```

A vector may use HTML characters, so you should also filter these. Add this rule:

```
$input= preg_replace('/(&#*w+)[x00-x20]+;/u', '$1;', $data);
$data = preg_replace('/(&#x*[0-9A-F]+);*/iu', '$1;', $input);
```

INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 3 ISSUE 1 (2017) PAGES 1648 - 1654
RECEIVED : 15.02.2017 PUBLISHED : 26.02.2017
February 26, 2017

But these alone are not going to help you. There are many places where input does not need script tags. An attacker can inject a few event functions to execute scripts. And there are many ways by which an attacker can bypass this filter.So, we need to think about all possibilities and add a few other things to make the filter stronger. And not only JavaScript, you also need to escape from cascading style sheets and XML data to prevent XSS

PHP AntiXSS

This is a nice PHP library that can help developers add an extra layer of protection from cross-site scripting vulnerabilities. It automatically detects the encoding of the data that must be filtered. Using of the library is easy.

HTML Purifier

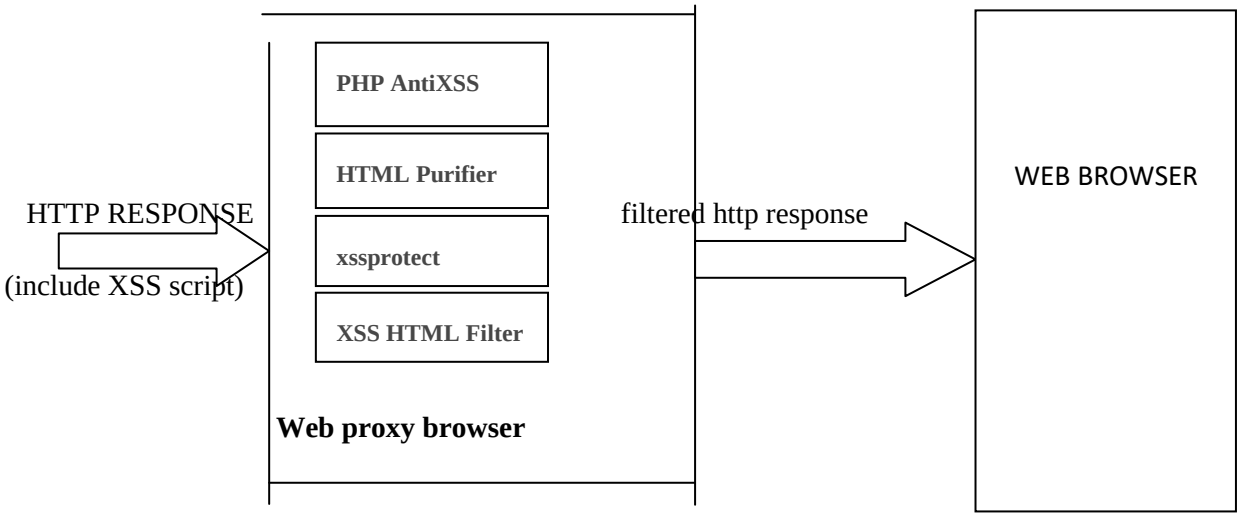
This is a standard HTML filtering library written in PHP. It removes all malicious code from the input and protects the website from XSS attack. It is also available as a plug-in for most PHP frameworks.

XSSprotect

xssprotect is another nice library that gives developers a way to clean XSS attack vectors. This Library works by creating the HTML tag tree of the webpage. Then it parses the page and matches all tags. After that, it calls the filter interface to filter improper HTML attributes and XSS attacks. This library is written in Java.

XSS HTML Filter

This is another XSS filter for Java. It is a simple single-class utility that can be used to properly sanitize user input against cross-site scripting and malicious HTML code injection.



Server side based XSS solution

INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 3 ISSUE 1 (2017) PAGES 1648 - 1654
RECEIVED : 15.02.2017 PUBLISHED : 26.02.2017
February 26, 2017

Use HTTPOnly cookie flag

According to the Microsoft Developer Network, HttpOnly is an additional flag included in a Set-Cookie HTTP response header. Using the HttpOnly flag when generating a cookie helps mitigate the risk of client side script accessing the protected cookie (if the browser supports it).

- The example below shows the syntax used within the **HTTP response header**:

```
Set-Cookie: <name>=<value>[; <Max-Age>=<age>]
[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; HttpOnly]
```

If the HttpOnly flag (optional) is included in the HTTP response header, the cookie cannot be accessed through client side script (again if the browser supports this flag). As a result, even if a cross-site scripting **(XSS)** flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser (primarily Internet Explorer) will not reveal the cookie to a third party. If a browser does not support HttpOnly and a website attempts to set an HttpOnly cookie, the HttpOnly flag will be ignored by the browser, thus creating a traditional, script accessible cookie. As a result, the cookie (typically your session cookie) becomes vulnerable to theft of modification by malicious script. Mitigating, [2]

1) Mitigating the Most Common XSS attack using HttpOnly

According to Michael Howard, Senior Security Program Manager in the Secure Windows Initiative group at Microsoft, the majority of XSS attacks target theft of session cookies. A server could help mitigate this issue by setting the HTTPOnly flag on a cookie it creates, indicating the cookie should not be accessible on the client. If a browser that supports HttpOnly detects a cookie containing the HttpOnly flag, and client side script code attempts to read the cookie, the browser returns an empty string as the result. This causes the attack to fail by preventing the malicious (usually XSS) code from sending the data to an attacker's website. Howard, [3]

Implement Content Security Policy

CSP is another layer of defense to help protect users from a variety of attack vectors such as XSS and other forms of content injection attacks. While it's not a silver bullet, it does help make it considerably more difficult for an attacker to inject content and exfiltrate data. Building websites securely is difficult. Even if you know general web security best practices it's still incredibly easy to overlook something or unwittingly introduce a security hole in an otherwise secure site.CSP works by restricting the origins that active and passive content can be loaded from. It can additionally restrict certain aspects of active content such as the execution of inline JavaScript, and the use of `eval()`.

Implementing CSP

To implement CSP, you must define lists of allowed origins for the all of the types of resources that your site utilizes. For example, if you have a simple site that needs to load scripts, stylesheets, and images hosted locally, as well as from the jQuery library from their CDN, you could go with:

```
Content-Security-Policy:
  default-src'self';
  script-src 'self' https://code.jquery.com;
```

The HTTP **X-XSS-Protection** response header is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks. Although these protections are largely unnecessary in modern browsers when sites implement a strong Content-Security-Policy that disables the use of inline JavaScript ('unsafe-inline'), they can still provide protections for users of older web browsers that don't yet support CSP.

CONCLUSION

Cross-site scripting is one of the most dangerous website vulnerabilities. It is used in various ways to harm website users. Mostly it is used to perform session hijacking attacks. We also know that patching XSS is possible but we can never be 100% sure that no one can break our filter. Hackers always find ways to break filter security. If you really want to make a hard-to-crack XSS filter, study most of the available XSS vectors. Then make a list of the different kinds of attack pattern. Analyze the list and code the functions to identify an attack pattern and block the attack. I have also added few open source available libraries that you can use if you do not know how to patch the vulnerability and secure your website.

REFERENCES

[1] K. Selvamani, A.Duraisamy and A.Kannan, "Protection of Web Applications from Cross-Site Scripting Attacks in Browser Side" (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 3, March 2010

[2] Imran Yusof and Al-Sakib Khan Pathan, "Preventing Persistent Cross-Site Scripting (XSS) Attack By Applying Pattern Filtering Approach".

[3] Monika Rohilla, Rakesh Kumar and Girdhar Gopal, "XSS Attacks: Analysis, Prevention & Detection", International Journal of Advanced Research in Computer Science and Software Engineering Volume 6, Issue 6, June 2016.

[4] Engin Kirda, Christopher Kruegel, Giovanni Vigna and Nenad Jovanovic, "Noxes: A Client-Side Solution for MitigatingCross-Site Scripting Attacks".

[5] Mike Ter Louw,V.N. Venkatakrishnan, "BLUEPRINT: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers", IEEE SYPOSIUM on security and privacy Oakland, 2009.

[6] Trevor Jim,Nikhil Swamy,Michael Hicks, "Defeating Script Injection Attacks with Browser-Enforced Embedded Policies", May 8–12, 2007, Banff, Alberta, Canada.ACM 978-1-59593-654-7/07/0005.

[7] Joaquin Garcia-Alfaro and Guillermo Navarro-Arribas, "Prevention of Cross-Site Scripting Attacks on Current Web Applications".