## CROSS SITE SCRIPTING (XSS)

“A web page contains both text and HTML mark-up that is generated by the server and interpreted by the client browser. Web sites that generate only static pages are able to have full control over how the browser interprets these pages. Web sites that generate dynamic pages do not have complete control over how their outputs are interpreted by the client.

The heart of the issue is that if mistrusted content can be introduced into a dynamic page, neither the web site nor the client has enough information to recognize that this has happened and take protective actions.”

XSS vulnerability arises when web applications take data from users and dynamically include it in Web pages without first properly validating the data.

**XSS is a security breach that takes advantage of dynamically generated Web pages. In an XSS attack, a Web application is sent with a script that activates when it is read by an unsuspecting user’s browser or by an application that has not protected itself against cross-site scripting. Because dynamic Web sites rely on user input, a malicious user can input malicious script into the page by hiding it within legitimate requests.**

Which Web pages are at risk?

Essentially, the problem affects dynamic page creation based on input that was not validated. Dynamically generated HTML pages can introduce security risks if inputs are not validated either on the way in or on the way out. Malicious script can be embedded within input that is submitted to Web pages and appear to browsers as originating from a trusted source.

If input to dynamic Web pages is not validated, we may encounter the following problems:

* Cookies can be set and read.
* User input can be intercepted.
* Malicious scripts can be executed by the client in the context of the trusted source.

Typical examples include the following types of Web pages:

* Search engines that return results pages based on user input.
* Login pages that store user accounts in databases, cookies, and so forth and later write the user name out to the client.
* Web forms that process credit card information.

Often attackers will inject JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable application to fool a user in order to gather data from them. Everything from account hijacking, changing of user settings, cookie theft/poisoning, or false advertising is possible.

Scripting tags that take advantage of XSS include <SCRIPT>, <OBJECT>, <APPLET>, <EMBED> and <FORM>. Common languages used for XSS include JavaScript, VBScript, HTML, ActiveX and Flash.

Nothing on the server is ever "infected".

This flaw can be found anywhere in an application where user input has been taken but not properly encoded. If the input is not properly encoded and sanitized, this injected malicious script will be sent to users. And a browser has no way to know that it should not trust a script. When the browser executes the script, a malicious action is performed on the client side. Most of the times, XSS is used to steal cookies and steal session tokens of a valid user to perform session hijacking.

Classification of XSS

**Reflected**[**Cross-site Scripting (XSS)**](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))**: Reflected XSS** are the most frequent type of **XSS** attacks found in the wild. **Reflected XSS** attacks are also known as non-persistent **XSS** attacks and, since the attack payload is delivered and executed via a single request and response, they are also referred to as first-order or type 1 **XSS.** A reflected attack is typically delivered via email or a neutral web site. The bait is an innocent-looking URL, pointing to a trusted site but containing the XSS vector. If the trusted site is vulnerable to the vector, clicking the link can cause the victim's browser to execute the injected script.

**Persistent (or stored) XSS**

The *persistent* (or *stored*) XSS vulnerability is a more devastating variant of a cross-site scripting flaw: it occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping. A classic example of this is with online message boards where users are allowed to post HTML formatted messages for other users to read.

**DOM XSS**

DOM Based XSS  (or as it is called in some texts, “type-0 XSS”) is an XSS attack wherein the attack payload is executed as a result of modifying the DOM “environment” in the victim’s browser used by the original client side script, so that the client side code runs in an “unexpected” manner. Among the objects in the DOM, there are several which the attacker can manipulate in order to generate the XSS condition, and the most popular, from this perspective, are the document.url, document.location and document.referrer objects.

**A few XSS examples*:***

**Example 1**

You see a search box on almost all websites. With the search box, you can search to find anything available on the website. On the search page where it shows search results, it also lists the search keyword in the form of “Search results for **Keyword**” or “You Searched for **Keyword**.”

Whatever a person searches for, it will be displayed on the web page along with search results. Now think what happens if an attacker tries to inject malicious script in place of ***Keyword***. Something like ***<script>alert(“XSS injection”)</script>***

If web application has not 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. It will be executed by the browser and it will display an alert box saying ***“XSS injection”***

**Example 2**

The comments section in blogs can be exploited if a user inputs a malicious script in place of comments. So whenever another ukj/ser comes to the comments page, the malicious script would be executed in the current user’s context.

### How to prevent XSS?

Essentially, prevention means that we follow good coding practice by running sanity checks on input to our routines/functions.

The following list outlines the general approaches to prevent cross-site scripting attacks:

* Encode output based on input parameters.
* Filter input parameters for special characters.
* Filter output based on input parameters for special characters.

There is a simple rule that we need to follow everywhere: Encode every datum that is given by a user. If data is 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.