**XSS REFLECTED**

"Cross-Site Scripting (XSS)" attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application using input from a user in the output, without validating or encoding it.An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the JavaScript. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

Because its a reflected XSS, the malicious code is not stored in the remote web application, so requires some social engineering (such as a link via email/chat).

**Description:** View any information that the user is able to view. Modify any information that the user is able to modify. Initiate interactions with other application users, including malicious attacks, that will appear to originate from the initial victim user.

**Objective:**

One way or another, steal the cookie of a logged in user.

**Impact:**A reflected XSS (or also called a non-persistent XSS attack) is a specific type of XSS whose malicious script bounces off of another website to the victim's browser. It is passed in the query, typically, in the URL. It makes exploitation as easy as tricking a user to click on a link.

**Prevention:**

* 1. Server-side

The best way to prevent cross-site scripting is to make sure that the web application does not make use of user input in the return HTML pages, without validating it first. Validation implies verification of the user input to determine if the input is valid, according to its purpose. In case the validation functions find script tags, either in plain text or encoded, they should sanitize the input before it is passed on to the response HTML and make sure that the script is rendered harmless.

* 1. Client side

Users should always be weary of what they click on; avoid playing seemingly harmless games, claiming random prizes or opening emails that don’t come from a trusted source. At the same time, users should avoid installing browser plugins which do not have a good reputation and those which are not really a necessity (such as toolbars), since these may make their browser vulnerable too.  Using secure and up-to-date web browsers will also help users keep away from “victim” status. Vulnerable websites are just the medium – the real target is you.

**LOW**

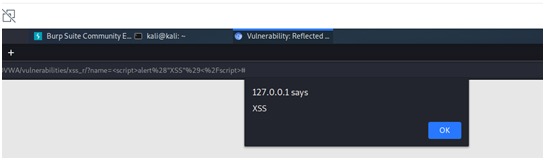
**Steps to reproduce:**

1. Configure your browser

2. Go to the dvwa page and set level of XSS(Reflected) to the low level

3. In the input box write “<script>alert(“XSS”)</script>”.

4. Now click on the submit button, got a pop up of alert 1.

5. For user cookies put this in input box “<script>alert(document.cookie)></script>”

**MEDIUM**

**Steps to reproduce:**

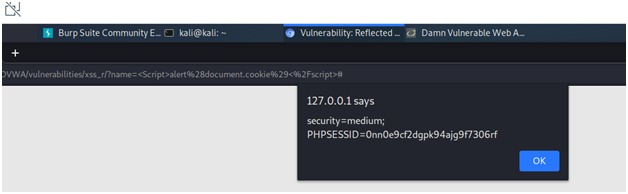
1.Configure your browser

2.Go to the dvwa page and set level of XSS(Reflected) to the medium level

3.In the input box write “<Script>alert(“XSS”)</script>”.

4.Now click on the submit button, got a pop up of alert 1.

5.For user cookies put this in input box “<Script>alert(document.cookie)></script>”



**HIGH**

**Steps to reproduce:**

1. Configure your browser

2. Go to the dvwa page and set level of XSS(Reflected) to the high level

3. In the input box write “<imgsrc=”” onerror=”alert(1)”>”.

4. Now click on the submit button, got a pop up of alert 1.

5. For user cookies put this in input box “<imgsrc=”” onerror=”alert(document.cookie)”>”

