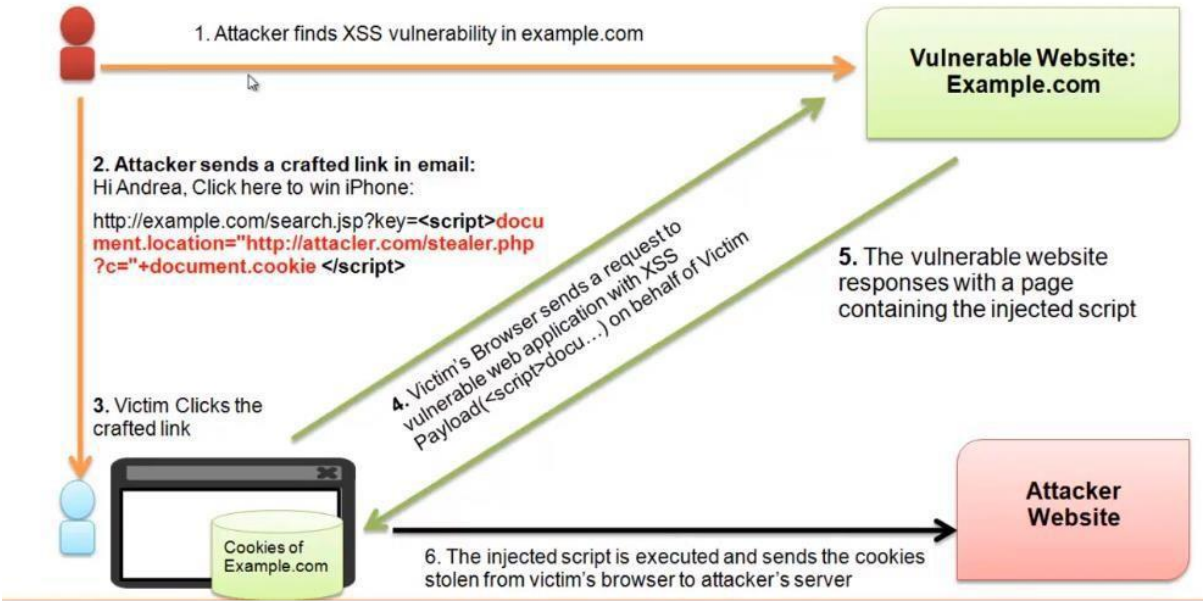
Secure Programming
XSS Lab

Introduction

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into 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 uses input from a user within the output it generates 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 script. 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 the browser and used with that site. These scripts can even rewrite the content of the HTML page.



Today's lab will look at Reflected XSS and Stored XSS. 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.

Stored or persistent XSS occurs when scripts are saved on a server and displayed as innocent links on pages, such as message boards. When users click on the malicious links, the scripts are executed.

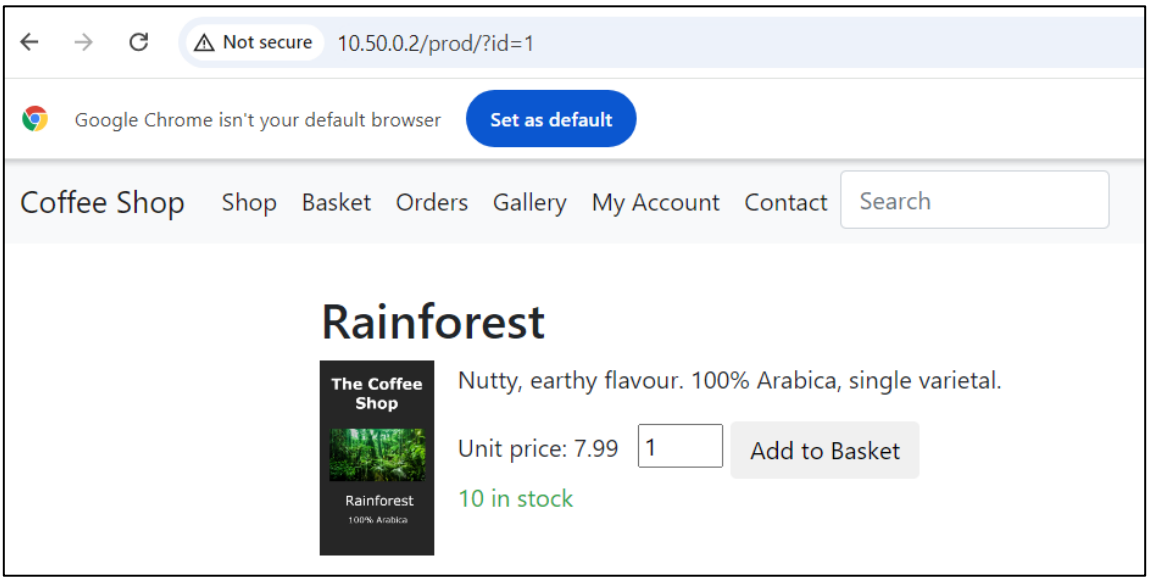
Exploiting a Reflected XSS Vulnerability

When trying to exploit reflected XSS, we are looking for places in the web application that user input is displayed back to the user, e.g., search boxes, error messages etc. The Coffeeshop application has a vulnerable page where the products are displayed, using the URL:

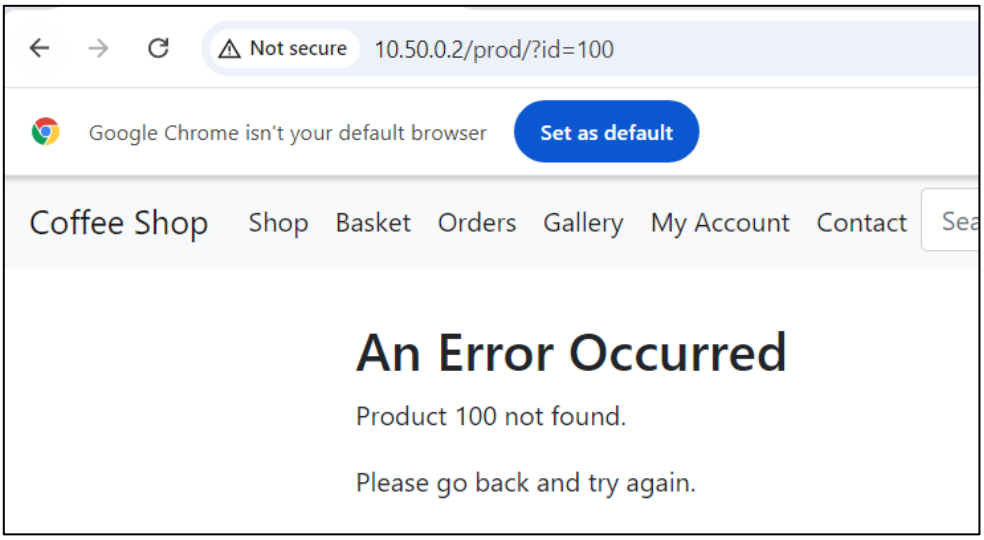
```
http://localhost:8080/prod/?id={productid}
```

If the URL is called with a valid product ID, for example

```
http:// localhost:8080/prod/?id=1
```



then the product is displayed. If an invalid ID is provided, it displays an error message stating that the requested ID was not found.



Since the user input is displayed unchanged, we can set a script into the URL and check to see if it works. Try the following:

```
http://localhost:8080/prod/?id=%3Cscript%3Ealert%28%22Hacked%22%29%3C%2Fscript
```

Q1. What do you see?
Q2. What do you think the inserted code means?

Change the attack URL to:

```
http://localhost:8080/prod/?id=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E
```

Q3. Now what do you see?

If this type of attack is to be successful, an attacker would have to craft a malicious email with the reflected XSS in a link and send it to a victim/ victims. This could be a simple "*You have won XXX, please click on link to claim your prize*" type lures or they could be more sophisticated. The JavaScript would not pop up an alert box but could redirect a victim's cookie to an attacker's server. Since the cookie is used for authentication, an attacker could then use the cookie to impersonate the victim and gain access to their account. We will look at an example of this in the next section.

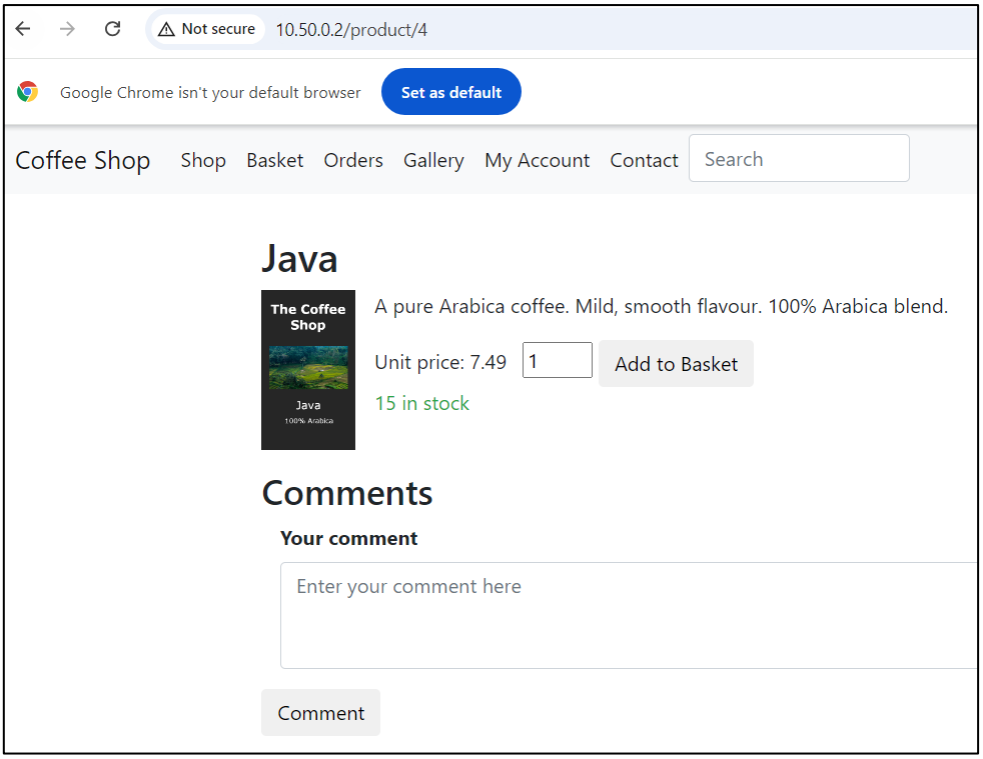
Exploiting a Stored XSS Vulnerability

Sometimes, user input is persisted in an application's database and then displayed in HTML pages. Examples include social media posts, forum posts, blog posts, product reviews etc. If an attacker can get malicious JavaScript code stored there, it will be executed whenever another user visits the page that displays that input. It will be executed as that user, with that user's cookies.

Our Coffeeshop application has one such vulnerability, in a form where users can leave comments about products. We will exploit such a vulnerability in this exercise.

Visit the Coffeeshop application at http://localhost:8080 in Chrome.

Log in as **bob** (see the file *coffeeshop/secrets/config.env* for the password). Now click on any of the products to see the details page. If you are logged in, you will see a box to enter a comment at the bottom of the page.



Enter the following comment and submit:

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

You should get a pop-up alert like the one below (if you don't, refresh the page). This alert contains 2 items: the CSRF token (more on that another week) and the session ID. The session ID is the authentication token for Bob's current session and this could be used by another user to impersonate Bob.