**Cross Site Scripting (XSS)**

It is an injection attack where malicious JavaScript gets injected into a web application with the intention of being executed by other users.

**Index**

[**Types of XSS** 2](#_Toc91702029)

[**Reflected XSS** 2](#_Toc91702030)

[**Stored XSS** 3](#_Toc91702031)

[**DOM XSS** 3](#_Toc91702032)

[**Blind XSS** 4](#_Toc91702033)

[**Practical Demonstration** 5](#_Toc91702034)

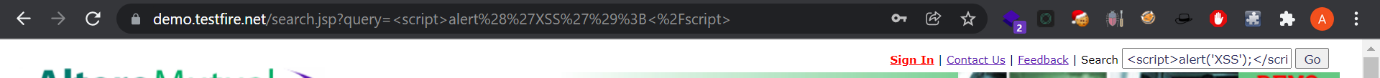
### **Types of XSS**

1. Reflected XSS
2. Stored XSS
3. DOM based XSS
4. Blind XSS

### **Reflected XSS**

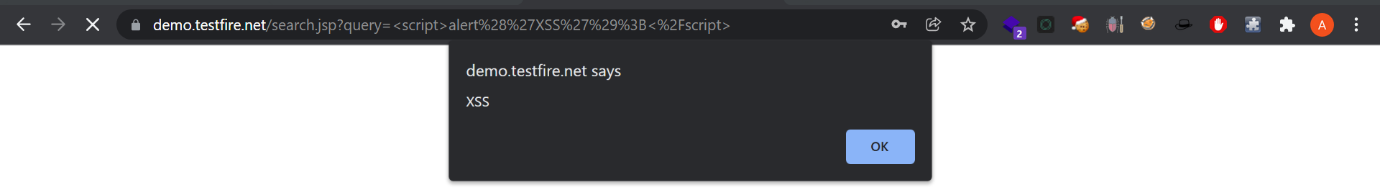
Reflected XSS happens when user-supplied data in an HTTP request is included in the webpage source without any validation.

For e.g.:



Input the payload in search bar: <script>alert('XSS');</script>

After clicking on Go button the below output is reflected



To test you need to look for every possible point of entry including:

* Parameters in URL
* URL file path
* Sometimes HTTP headers

### **Stored XSS**

As the name tells, the payload gets stored on web application as data and gets executed whenever the data is called.

To test you'll need to look every possible point of entry where it seems data is stored and then shown back in areas that other users have access to. For e.g.;

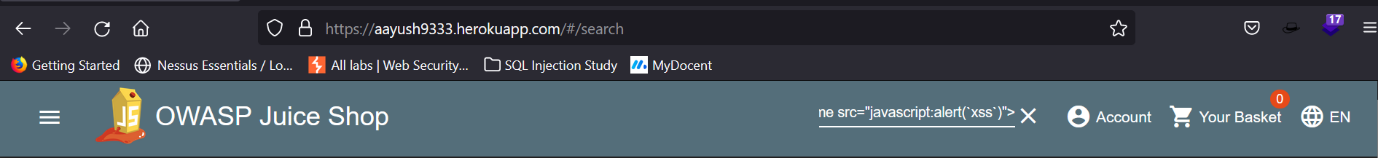
* Comments on a blog
* User profile information
* Website Listings

### **DOM XSS**

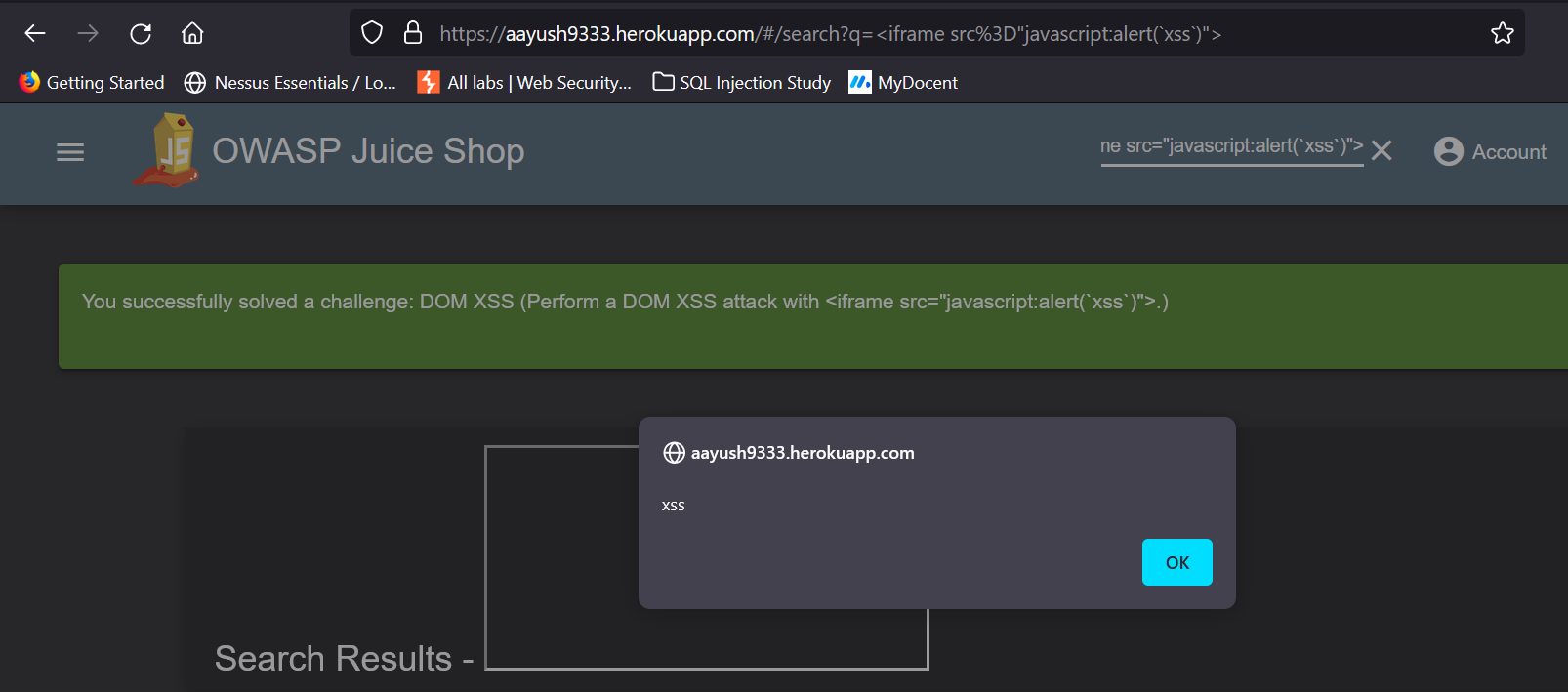
DOM stands for Document Object Model and is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style and content. A web page is a document, and this document can be either displayed in the browser window or as the HTML source.

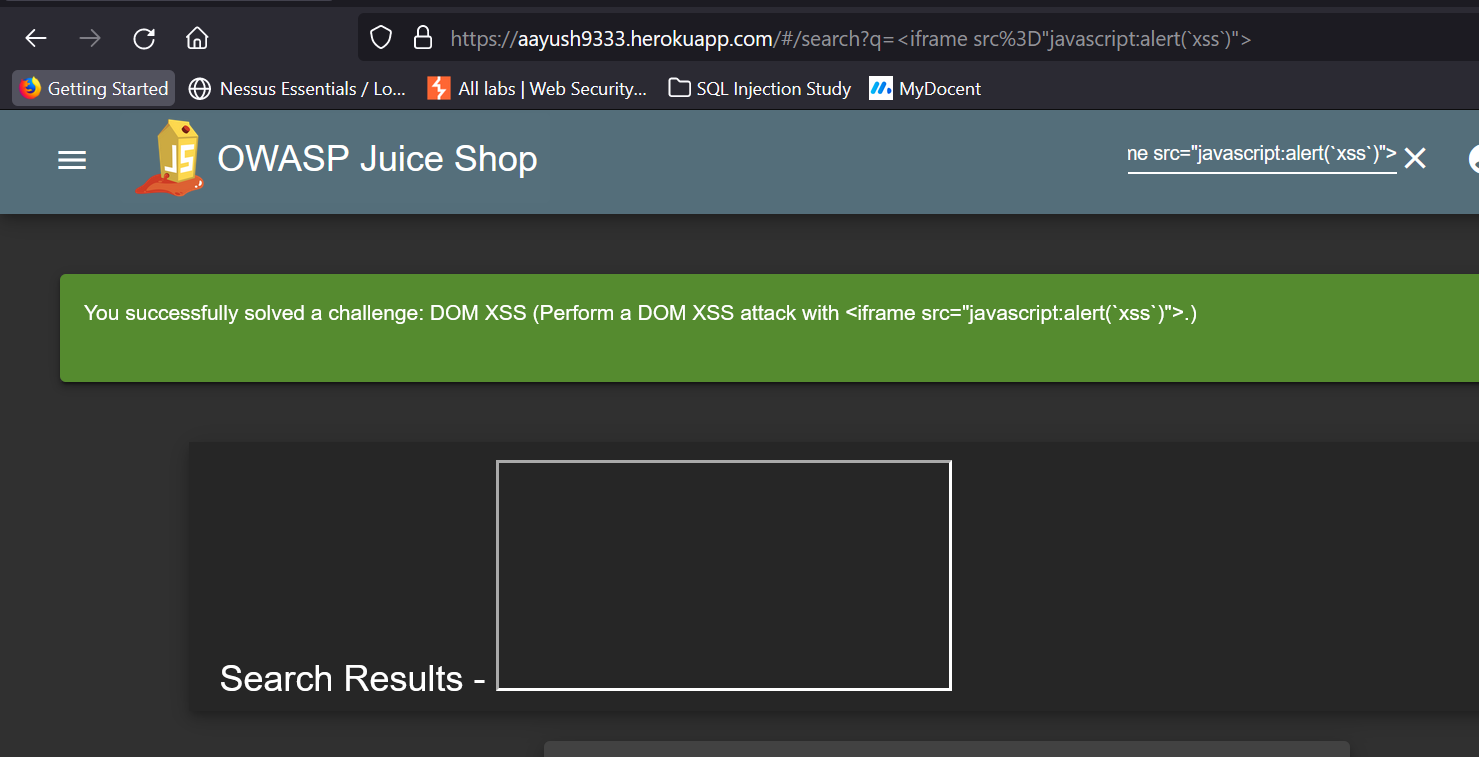
For e.g.:

In the search bar of below webpage, we used the payload: <iframe src="javascript:alert(`xss`)">



After pressing enter the alert raised and behind that alert there is a frame box which is part of website’s DOM.





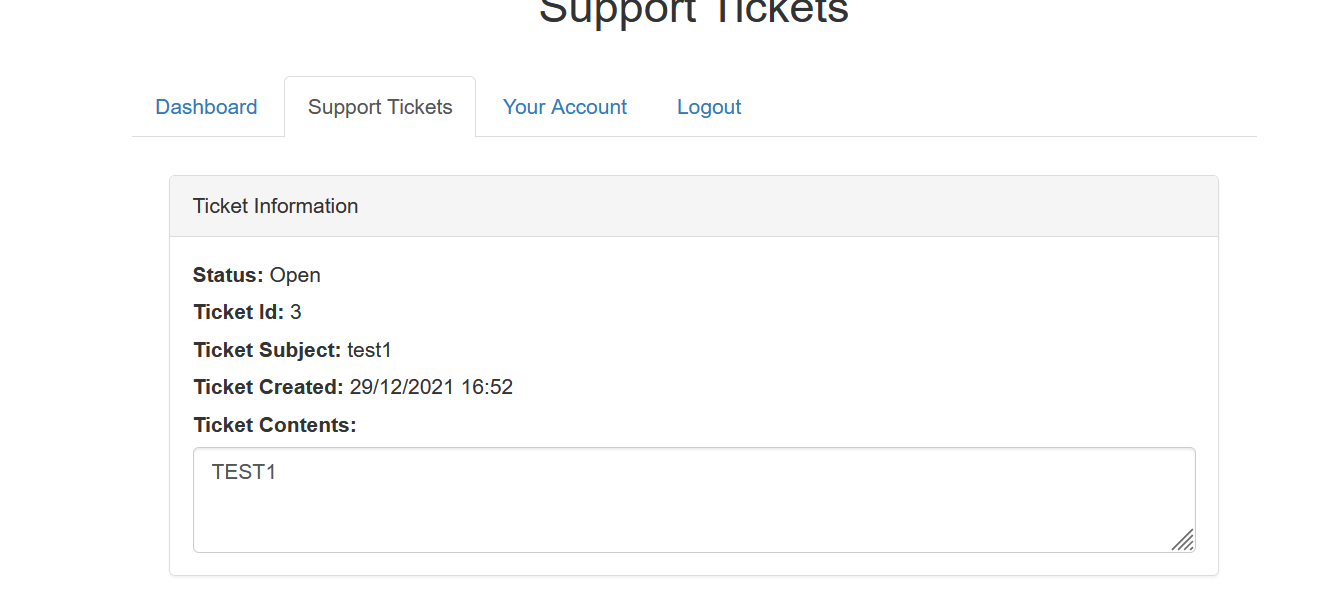
### **Blind XSS**

Blind XSS is similar to a stored XSS in which payload gets stored on the website for another user to view, but in this instance, you can't see the payload working or be able to test it against yourself first.

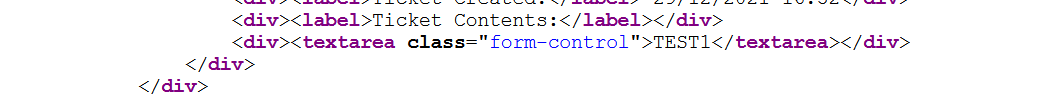
### **Practical Demonstration**

Using Stored XSS

Data is stored in below form

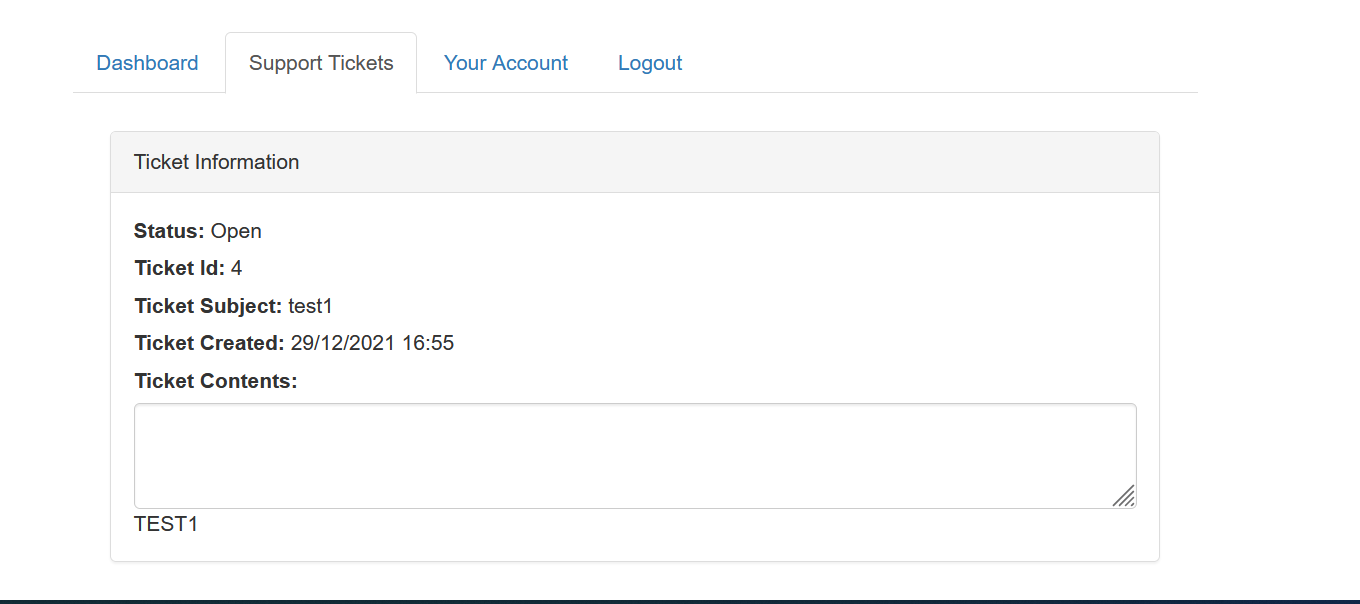


The below code shows the data being stored

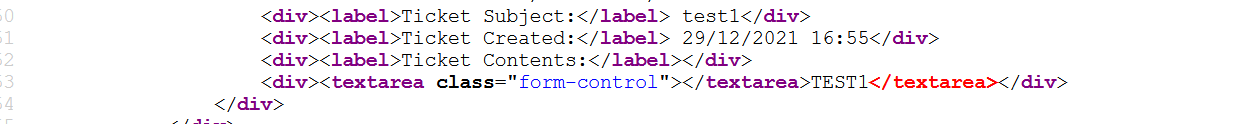


Let us modify the input to </textarea>TEST1 and submit

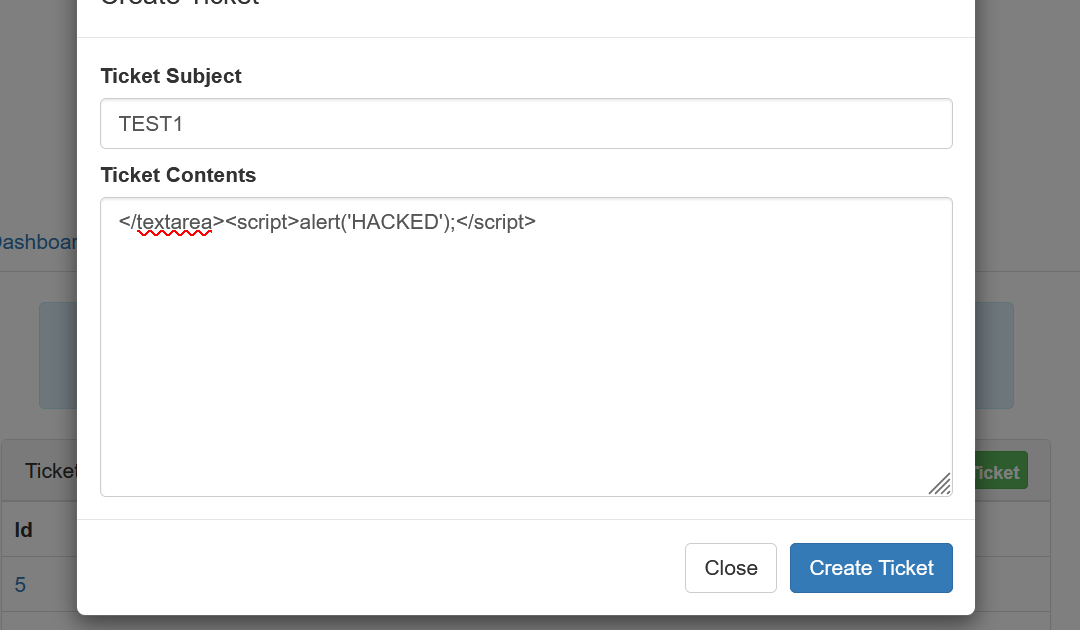
It will get stored in below format



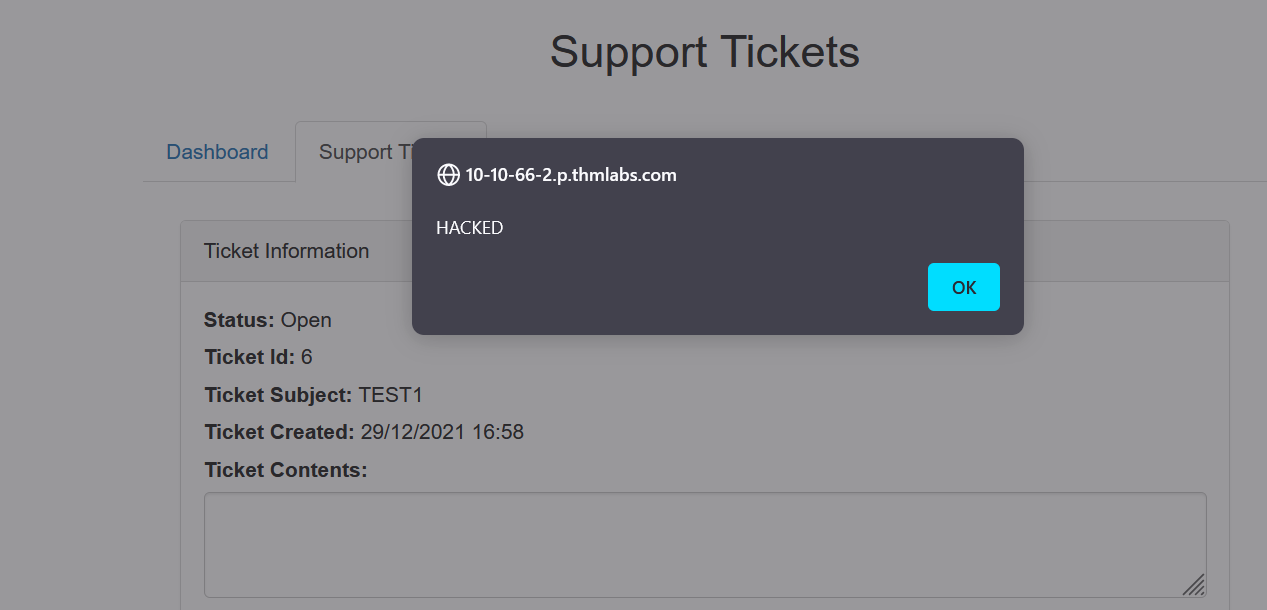
Executing through below code



Now let’s input malicious code:



It will get stored on database and on reloading the page we will get this alert box:



**Thus application is vulnerable.**