

## CMPE-279 Assignment-3

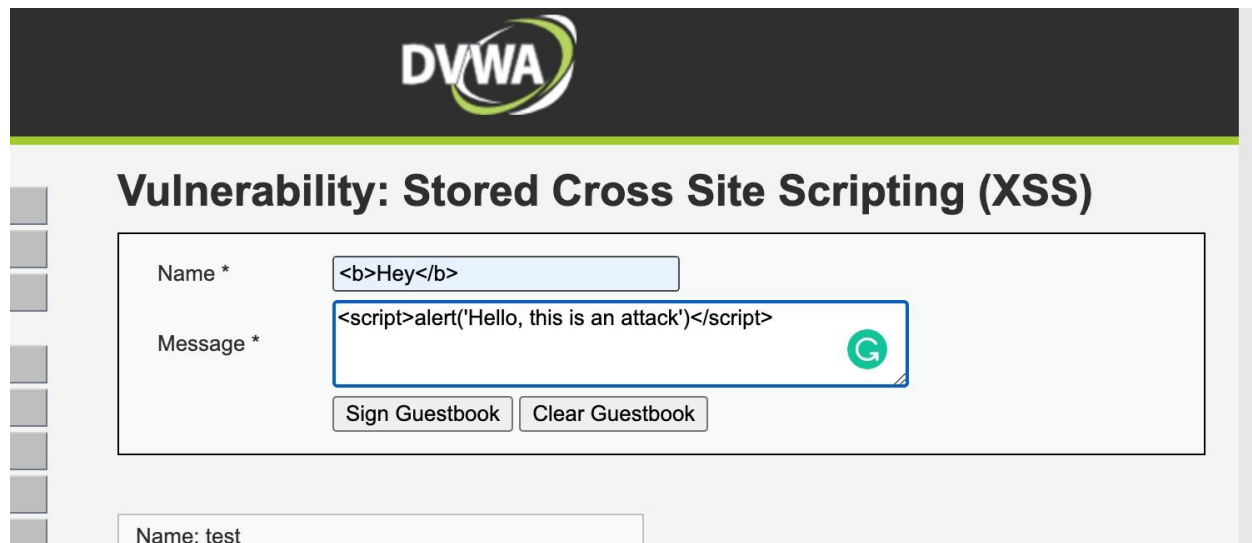
**Ayush Gupta(Sjsu id: 014952184)**

**Chetan Nain(Sjsu id: 015761122)**

### Q.1 Describe the attack you used. How did it work?

Ans. We used Stored XSS attack for this assignment. Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

To perform this attack we navigated to XSS stored tab, and entered Html tags in name and message field. We used `<b>Hey</b>` as name and `<script>alert('Hello, this is an attack')</script>` as message field. In response we saw alert message on the screen. On every refresh to the page, we see the same alert without anything malicious in the url. This is due to the fact that our script was stored in some permanent storage.



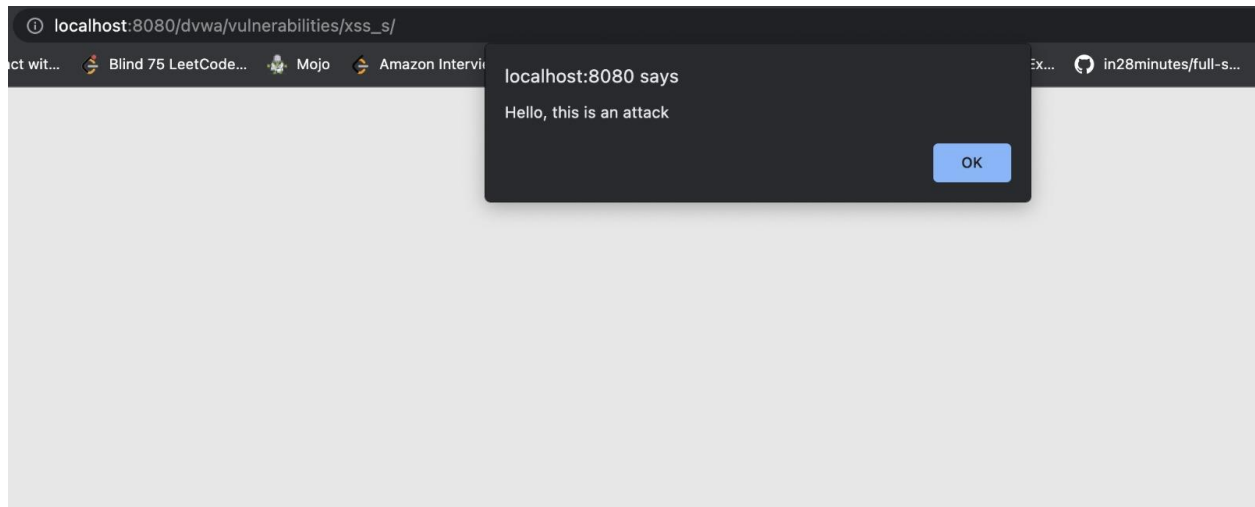
**DVWA**

### Vulnerability: Stored Cross Site Scripting (XSS)

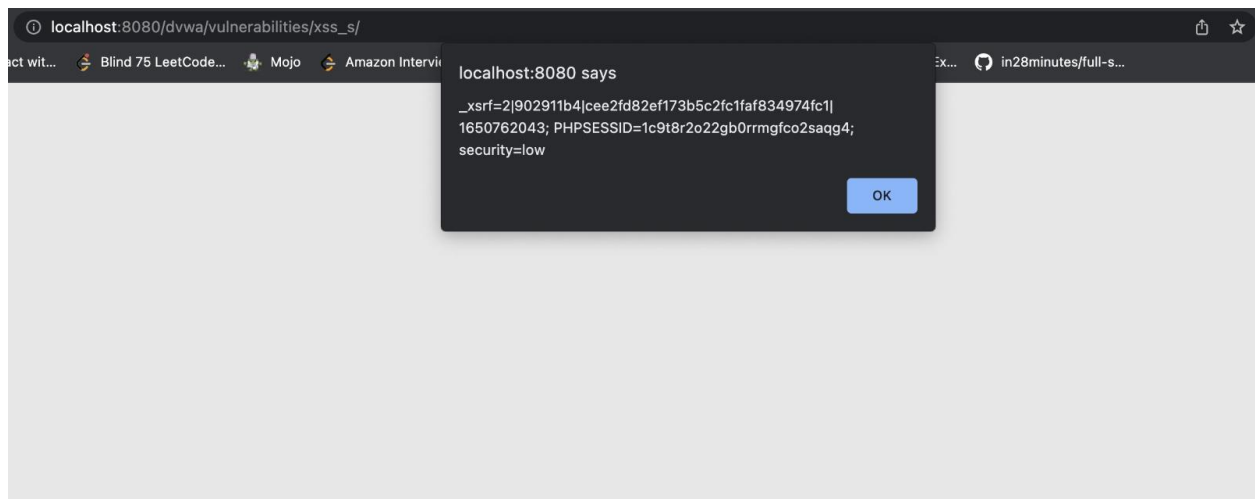
Name \*

Message \*

Name: test



Next, we tried to get session ID and cookie to display using `<script>alert(document.cookie)</script>` as message. Below is the screenshot showing the session id.



## Q.2 Does your attack work in “Medium” security level?

Ans. No, when we tried to use same name and message input to get alert, It didnt work out. When we inspected the code, we thought that there was some checker which strips the HTML tags from the input. But We were able to introduce an attack by changing the input size of Name field using the source tab. It was set to 10 initially but we changed it to 100. Now we were able to input longer Name input.

**Q.3 Set the security mode to “Low” and examine the code that is vulnerable, and then set the security mode to “High” and reexamine the same code. What changed? How do the changes prevent the attack from succeeding?**

In low security level, we were able to render alert message using `<script>alert(“Hello, this is attack”)</script>`. when we tried the same thing with High security level, we were not able to reproduce the result.

On further Analysis, we understood that, While in low security level, `stripslashes()` is used for sanity check on message field and there is no sanitation checks on name fields. `stripslashes()` removes any back slashes input has along with quotes in the input string. But we were able to inject HTML tags and javascript commands in low security level.

In High level in backend there must be some input sanitisation or html encoding on the message field when we give user input. Like `strip_tags()` is used in high security, which strips all the HTML tags from the input. Input is sanitized using `strip_tags(addslashes($message))` and `htmlspecialchars()`. `addslashes()` d any backslashes to escape character sequence and `strip_tags()` is applied to the output which strips all the null characters. After this sanitization, input is passed to `htmlspecialchars()` which detects special character like `<`, `>` etc.