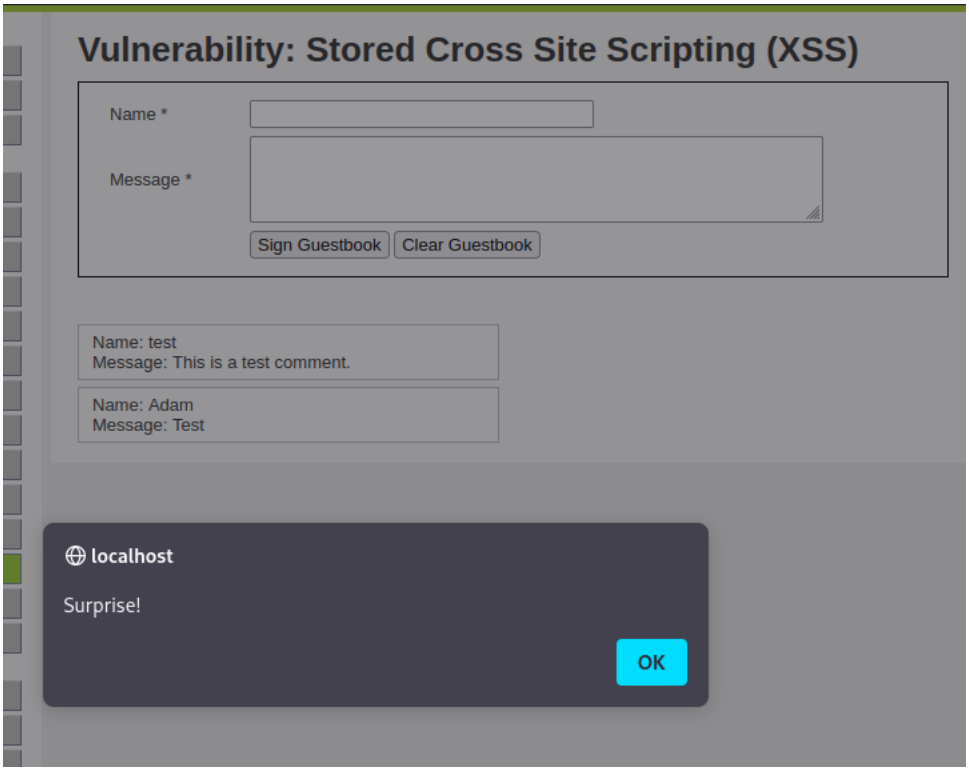
Adam Romayor

Assignment 4

Describe the attack you used. How did it work?

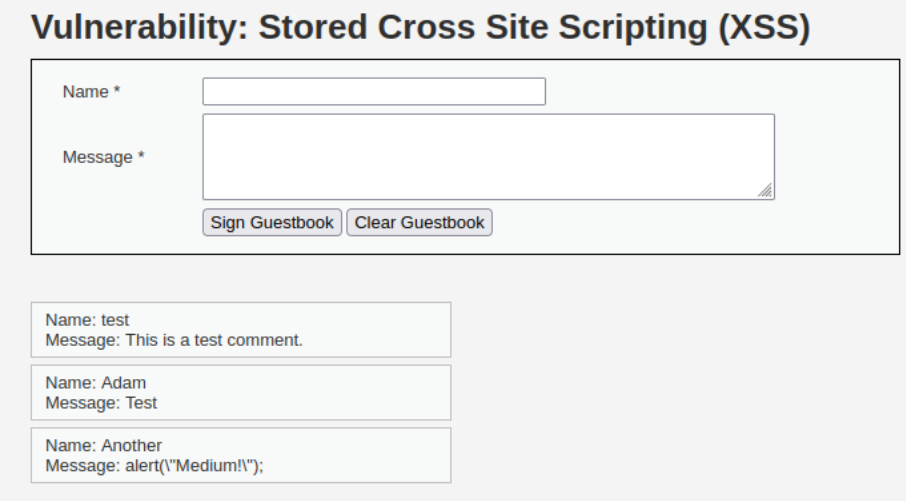
For this assignment I performed a stored XSS attack against DVWA. In the comment box I included the following string: `Test<script>alert("Surprise!");</script>`

This comment was stored in a database. When all the comments are displayed on the browser the code inside the script tags is executed as real code, so an alert pops up. If the user clicks to a different tab, and then clicks back to the XSS (Stored) tab, the alert will pop up again, because it executes the code every time the page is rendered.



Does your attack work in “Medium” security level?

Switching it to “Medium” security gave interesting results. The old exploit, when it was set to “Low” was still in the database, so I still get the same alert popup. However, when I create another comment using the input string: `<script>alert("Medium!");</script>` it does not create a second alert. This exploit prints the comment without the script tags and has escape characters before the quotation marks. This tells me that there was some initial processing before it was stored in the database, but there is no additional security before displaying the comments.



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?

The first line changed sanitizes the message input. In the low security code, it uses the function stripslashes(). In the high security code, it uses the functions addslashes() and strip_tags(). The first function, addslashes, escapes all single quotes, double quotes, backlashes, and NUL bytes. This prevents an attacker from doing an exploit that requires quotations like the code alert(“Hello”). Since quotes are escaped, this line would be printed as alert(\“Hello\”). The strip_tags function removes all HTML tags in the input message. This means any <script> tags are removed before it gets stored into the database. When the comments are received from the database, they don’t include any script tags that would potentially cause the browser to execute injected code. The final change in high security code adds a line to sanitize name input. It uses the function preg_replace() to replace any script tag with an empty string. This prevents an attacker from injecting code into the name input field.