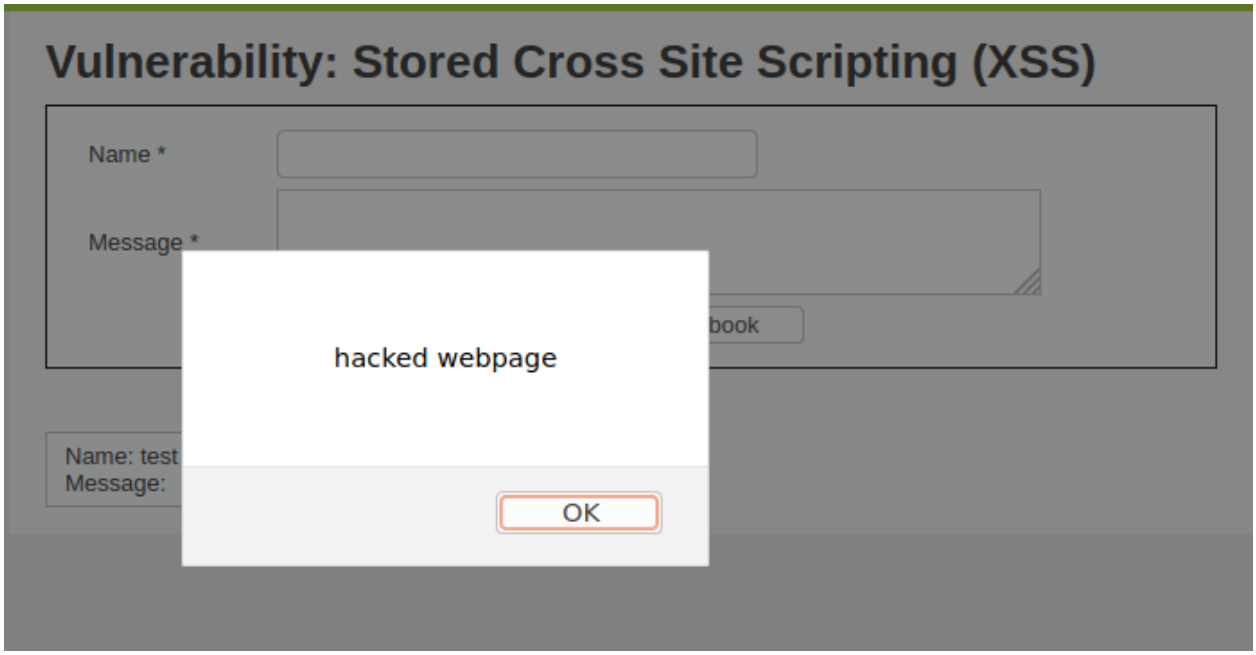
CMPE 279 Assignment 4

Ashwin Ramaswamy
Vandana Chandola

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

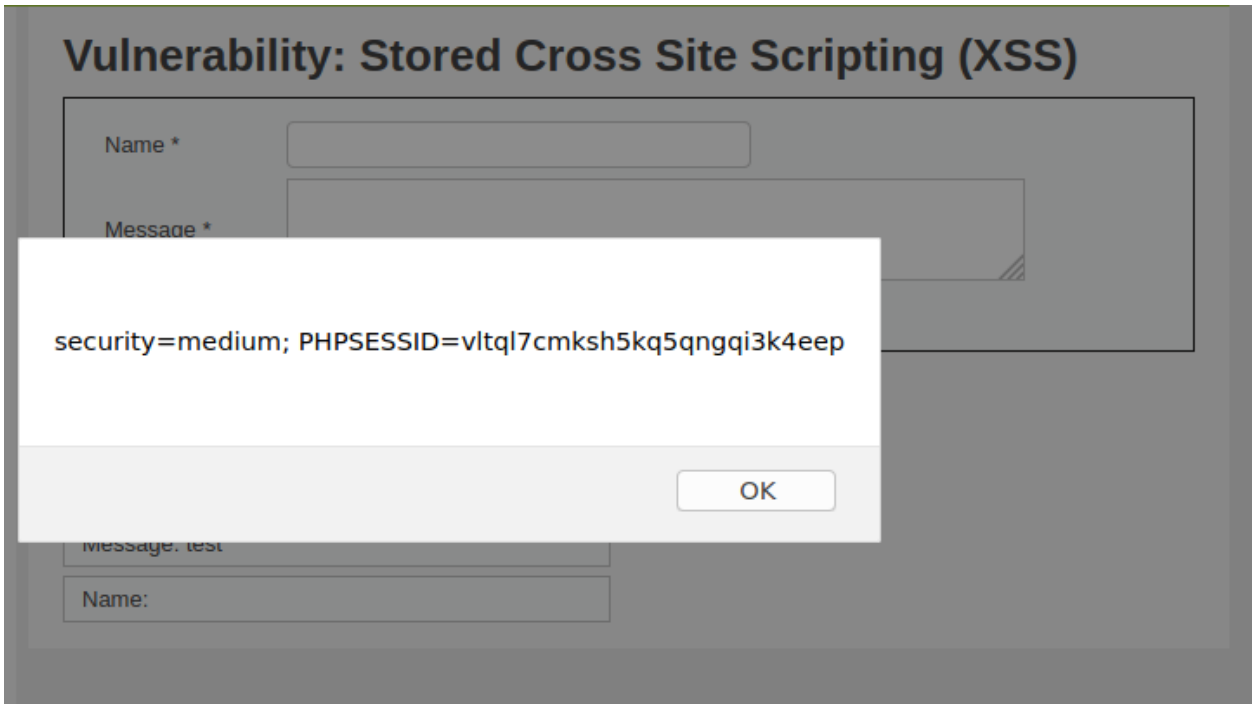
We performed a stored XSS attack. To do so, in the guestbook we set the “Name” field to any arbitrary text, but then we set the “Message” field to: **<script>alert(“hacked webpage”)</script>**. Upon submitting this entry, we receive a popup alert with the exact alert message we typed in. Additionally, whenever we leave the “XSS stored” page or refresh the page, we see this popup without needing to type anything. The message in the alert can even be “**document.cookie**” to get critical information every time a user enters that website.



2) Does your attack work in “Medium” security level?

For a “Medium” security level, the previous method of the stored XSS did not work because the source code strips tags from the “Message” field. However, we are able to take advantage of this by writing the desired command in the “Name” input field, which is less restrictive when sanitizing input text than the “Message” input field. But the “Name” field has a maxLength attribute set to 10, which can be removed in the browser’s dev tools or by inspecting and editing the source code in the browser. The

“Name” field just replaces the literal character sequence “**<script>**” with an empty string, so we can write the command “**<SCRIPT>alert(document.cookie)</SCRIPT>**” into the payload and achieve the intended result. Now every time a user visits the webpage, they will see the session cookie as such:



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?

The “Low” security mode sanitizes the message input by applying the “stripslashes()” PHP function and checks for escape strings. This can be used to unquote a quoted string and strip the backslashes off of a string. This is impractical for security purposes because it has no mechanism to protect from a user inserting JavaScript commands with a “<script>” tag. The name input field does not sanitize anything. The “High” security mode sanitizes the input with the PHP function “addslashes()”, which adds backslashes to all characters that need to be escaped (single quotes, double quotes, backslashes, NUL). Then it applies the function “strip_tags()” to this result which returns a string with all NULL bytes, HTML and PHP tags stripped from the string. Finally it applies the function “htmlspecialchars()” to the result to convert characters like “<” and “>” to html encoding, or literal text. This security mode also sanitizes the “Name” input field by using regex to search for any sequences that have the characters “s”, “c”, “r”, “i”, “p”, and “t” within brackets even if there are characters between them. It is then

replaced with an empty string. This field prevents the attack from working by eliminating the ability of escape characters to hide JavaScript functionality from regular text input fields. This means that special characters, script tags, and even characters hidden around other characters are parsed and either eliminated, or converted into their HTML encoding and processed directly as text.