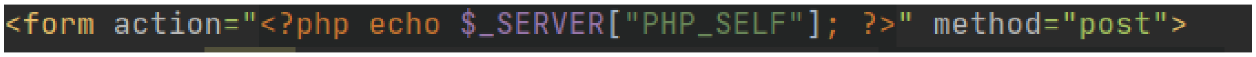
PHP Form Validation

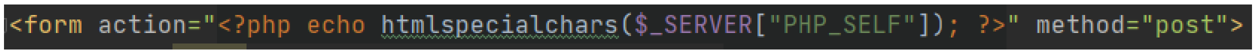
In this lab, you will improve the functionality and security of the form created in the previous lab.

1. Create file named **lab07.php**.
2. Copy all your code from lab06.php and paste it into lab07.php, making sure to change the page title and any supporting comments.
3. First, you will fix a potential exploit that was introduced in the last lab with the code below.



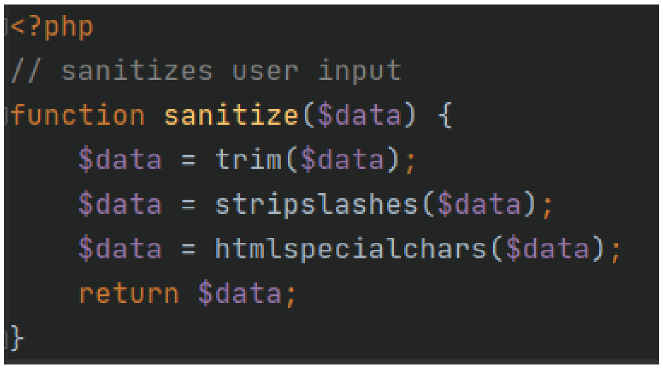
```
<form action="<?php echo $_SERVER["PHP_SELF"]; ?>" method="post">
```

4. Using $_SERVER["PHP_SELF"] without sanitizing its value can allow hackers to perform a cross-site scripting (XSS) attack to inject client-side script (JavaScript) into web pages visited by other users. Follow the link and read the section Big Note on PHP Form Security.
5. You can test this attack by editing this URL to point to your script and executing it in a browser.
   a. **http://csphplinux.northeaststate.edu/~yourusername/lab06.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E**
   b. Or by clicking the following link to the instructor's lab06.php. Warning: only follow the link if it looks like the URL above. If you have not received this file from the instructor directly, do not click the link above without first inspecting it. Using any exploits outlined in class for any purpose other than educational demonstrations is unethical and potentially illegal. XSS Example for Security Demonstration Purposes Only.
6. You will fix this exploit using the **htmlspecialchars()** function, which will replace HTML characters like **<** and **>** with **&lt;** and **&gt;** . This will prevent malicious client-side scripts from executing.

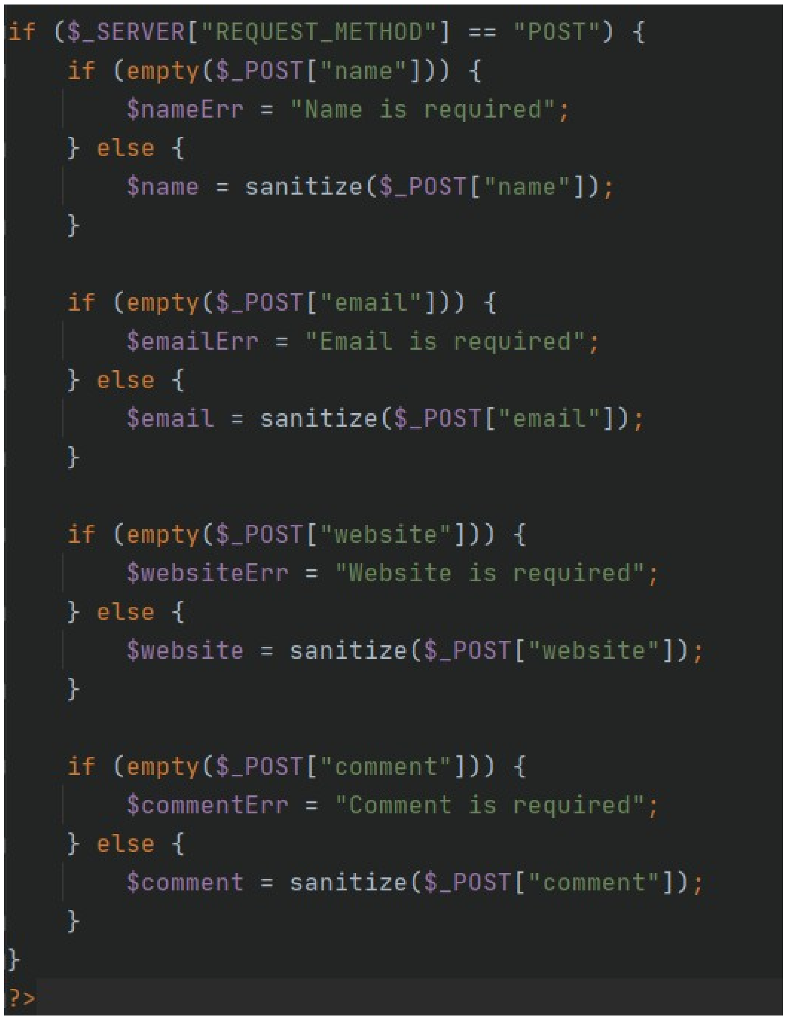


```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post">
```

7. The **htmlspecialchars()** function is one of several functions that can help you sanitize your user input forms and improve your web application's security. It is vital to always question the validity of user input. Sanitizing user input is a common practice in PHP, so next you will develop a function that you can use to methodically secure user input.
8. At the top of your first PHP block, add a method called **sanitize()**. Then call the sanitize method each time you access values that have been set by the user in the $_POST superglobal. Your code should look the examples below.



```php
<?php
// sanitizes user input
function sanitize($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

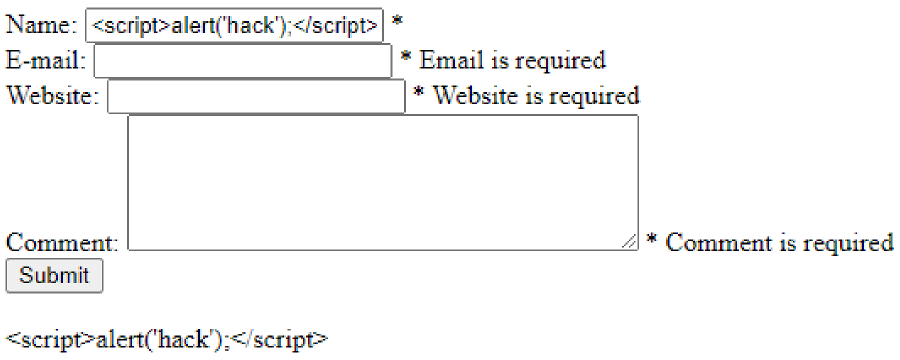


```php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else {
        $name = sanitize($_POST["name"]);
    }

    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = sanitize($_POST["email"]);
    }

    if (empty($_POST["website"])) {
        $websiteErr = "Website is required";
    } else {
        $website = sanitize($_POST["website"]);
    }

    if (empty($_POST["comment"])) {
        $commentErr = "Comment is required";
    } else {
        $comment = sanitize($_POST["comment"]);
    }
}
?>
```

9. The PHP documentation on the sanitization functions can be found at the links below.
    a. trim()
    b. stripslashes()

   c. htmlspecialchars()

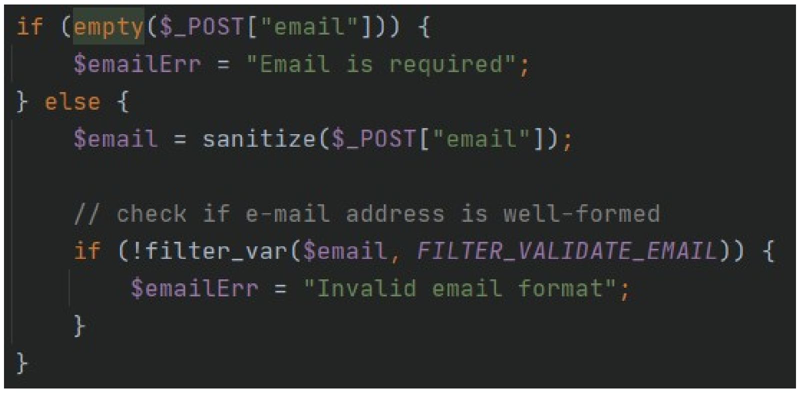
10. You can test your sanitize function by adding extra white space or backslashes to input. Or with something like below. View the source to see the effect of htmlspecialchars();

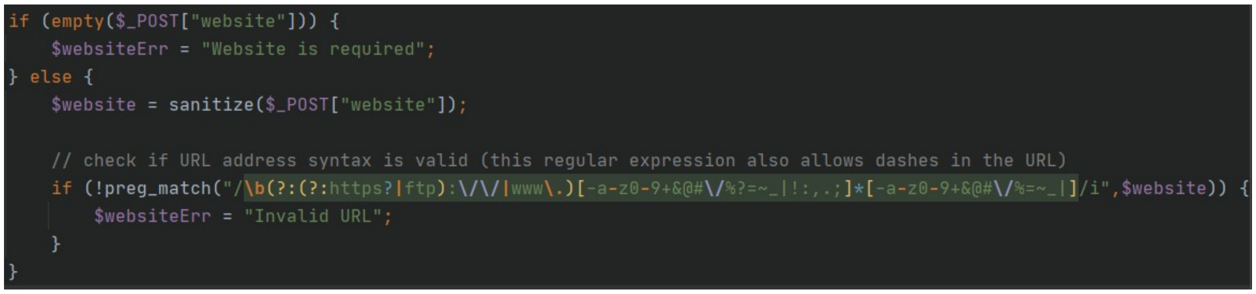

Name: <script>alert('hack');</script> *
E-mail: * Email is required
Website: * Website is required
Comment: * Comment is required
Submit

<script>alert('hack');</script>

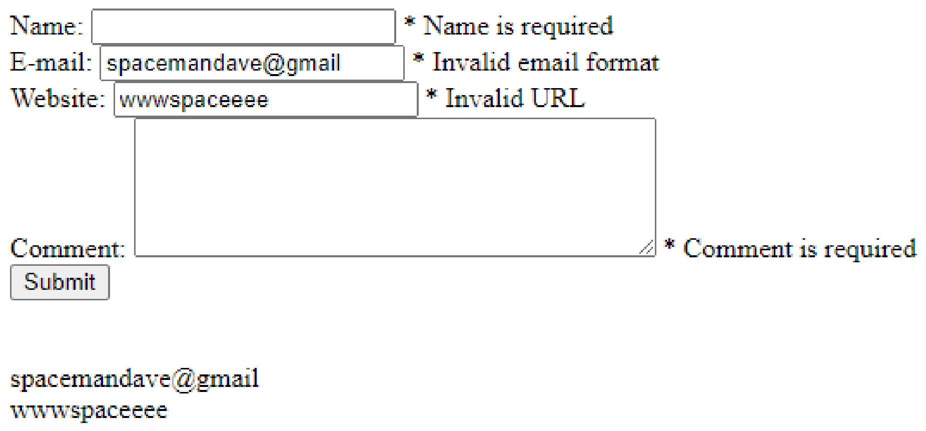
```
<br>&lt;script&gt;alert('hack');&lt;/script&gt;<br><br><br><br>
```

11. Not all user input is malicious, thankfully, but you still may need to validate user input to make sure it is of the correct form. For this step use the code example found here to see how to update your email and website processing code to include email and URL validation.


```
if (empty($_POST["email"])) {
    $emailErr = "Email is required";
} else {
    $email = sanitize($_POST["email"]);

    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailErr = "Invalid email format";
    }
}
```


```
if (empty($_POST["website"])) {
    $websiteErr = "Website is required";
} else {
    $website = sanitize($_POST["website"]);

    // check if URL address syntax is valid (this regular expression also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
        $websiteErr = "Invalid URL";
    }
}
```

12. After making these changes your form script should alert the user of invalid input for email and website.



```
Name: [                    ] * Name is required
E-mail: [spacemandave@gmail] * Invalid email format
Website: [wwwspaceeee      ] * Invalid URL
Comment: [                 ] * Comment is required
[Submit]

spacemandave@gmail
wwwspaceeee
```

Deliverables

1. Demonstrate the code and execution to the instructor during this lab, during office hours, or during the next lab period.
2. Upload the instructor approved .php file (source code only) to the Lab 7 D2L drop box.