
Cross Site Scripting (XSS)
</script>
</>
HACKED!
<script>alert('Hacked!')</script>

Lab - Cross Site Scripting

Objectives

In this lab, you will perform Reflected XSS and Stored XSS attacks against the DVWA (Damn Vulnerable Web Application!) at low, medium, and high security levels.

- Part 1: Perform Reflective Cross Site Scripting Exploits
- Part 2: Perform Stored Cross Site Scripting Exploits

Background / Scenario

In this lab, you will perform penetration tests of a web application to determine if it has been securely designed.

DVWA (Damn Vulnerable Web Application!) is a PHP/MySQL web application. It is designed to be vulnerable to common attacks to allow security professionals to test their skills and tools and students and teachers to learn and understand web application security in a legal environment.

DVWA provides four levels of security: Low, Medium, High, and Impossible. Each security level requires different skills to perform exploits. The security levels reflect different levels of security that developers may code into their applications. In this lab, you will perform exploits against three of the security levels, Low, Medium, and High, allowing you to adjust your attacks to compromise them.

Required Resources

- Kali VM customized for the Ethical Hacker course
- Internet access

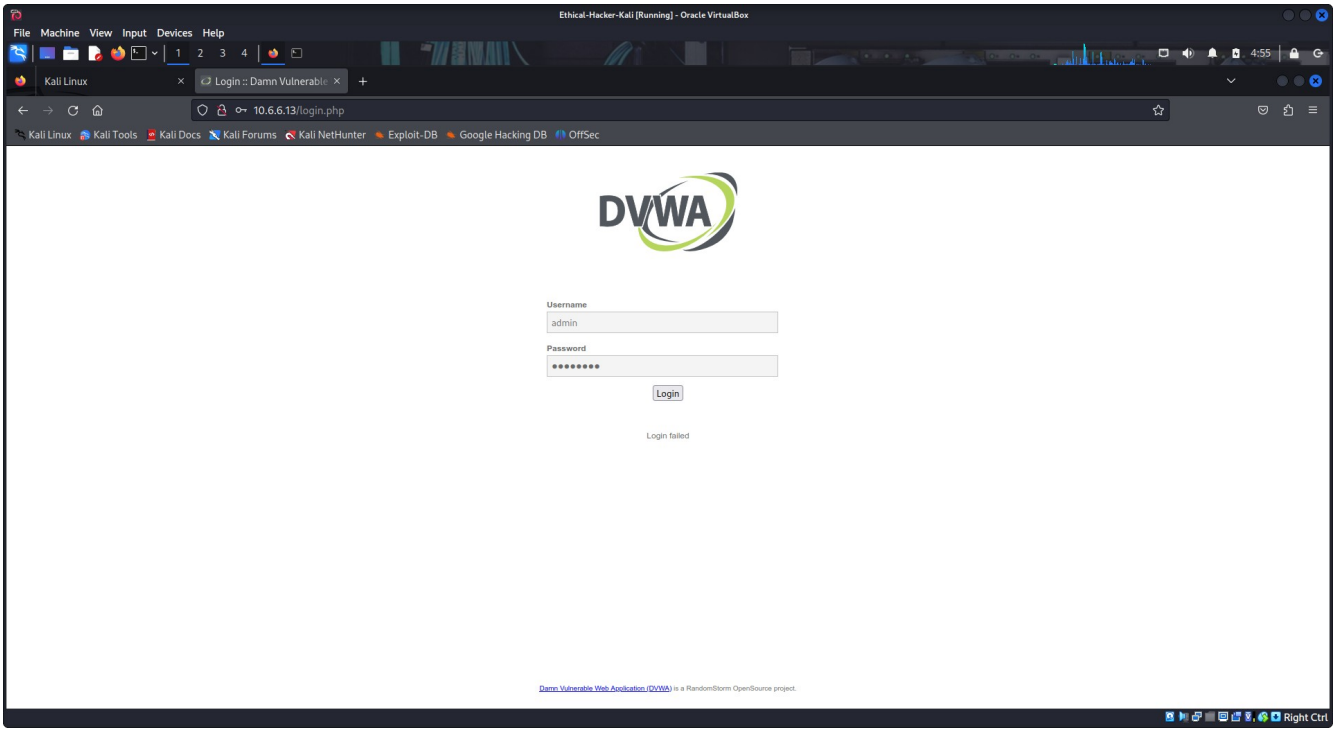
Instructions

Part 1: Perform Reflective Cross Site Scripting Exploits

A **Reflected XSS attack** is one in which a malicious script is reflected off a web server to the user's browser. The script is activated through a link that the victim clicks. This will send a request to the website that has a vulnerability that enables execution of the malicious script.

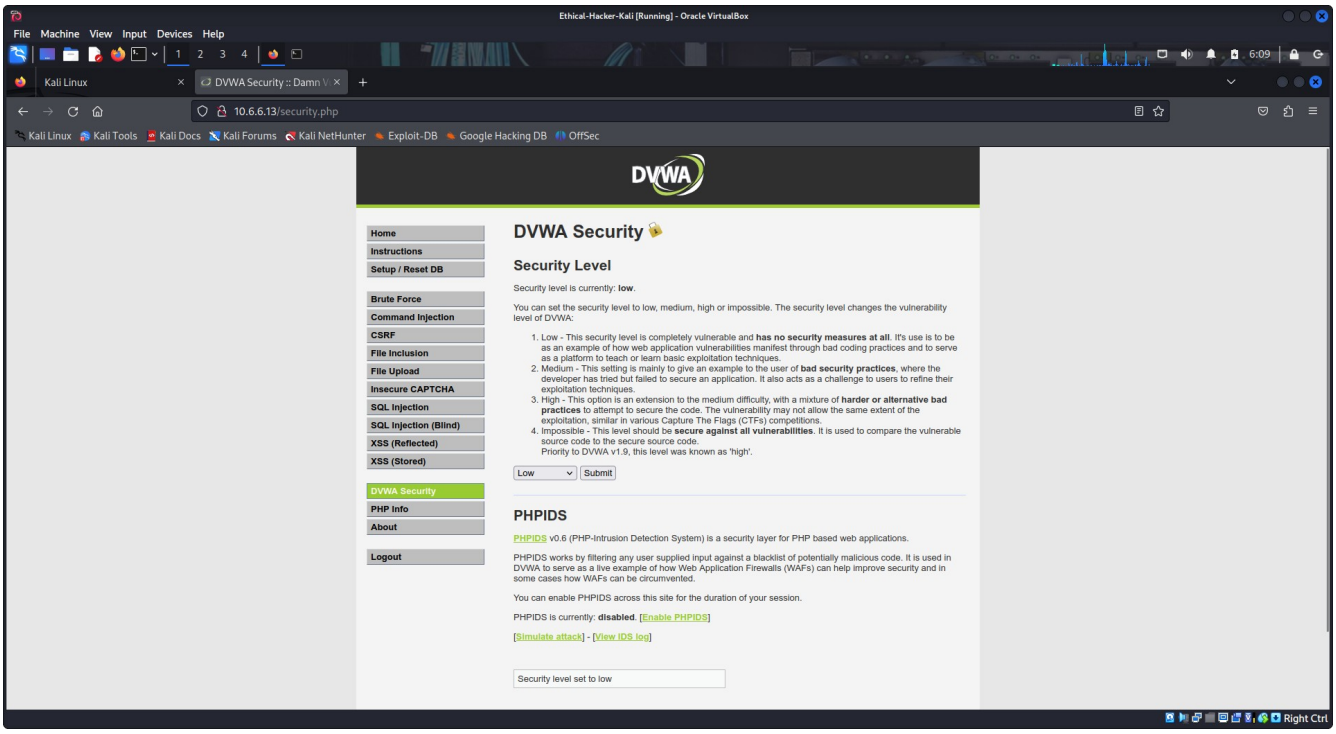
Step 1: Log into DVWA.

a. From the Kali Linux VM, open a browser and navigate to the DVWA application.
b. Enter the following URL into the browser http://10.6.6.13.
c. At the login prompt, enter the credentials: **admin**/**password**.



Step 2: Perform a Reflected XSS attack at Low security level.

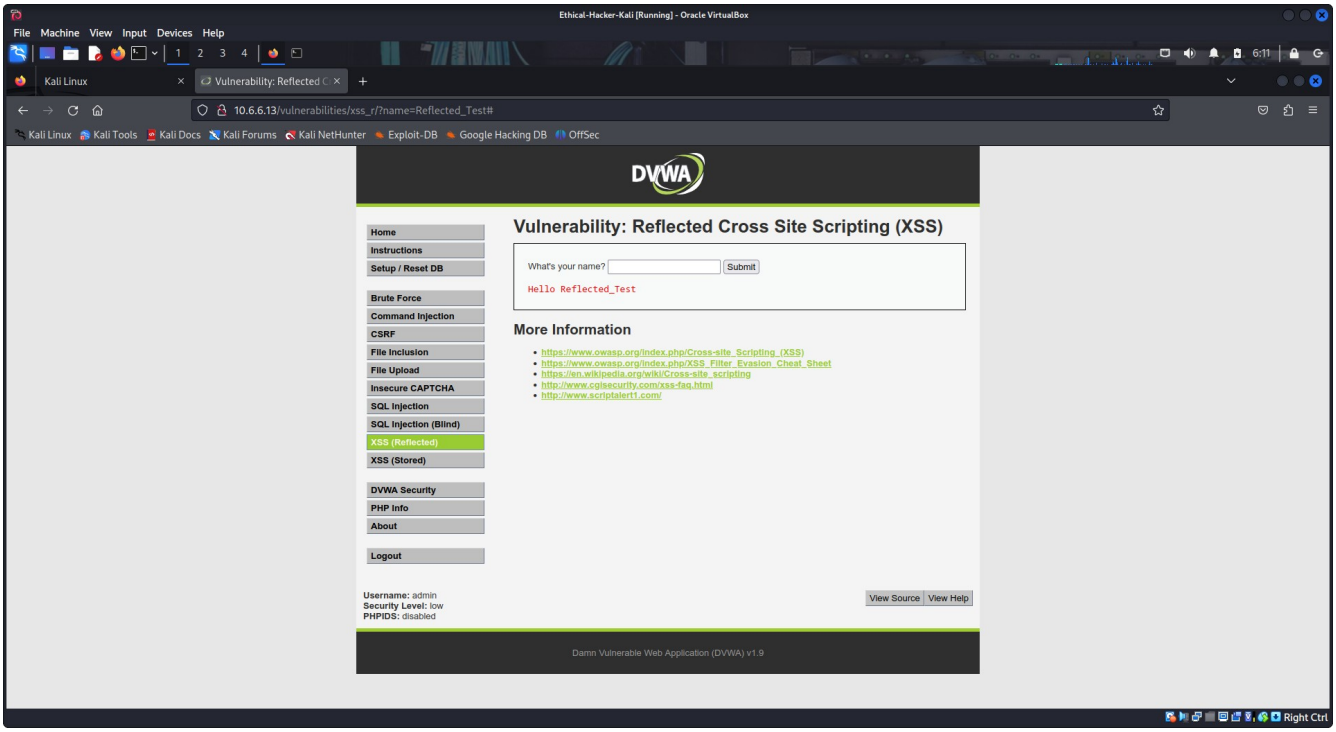
a. Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**.



b. Select **XSS (Reflected)** from the left menu.

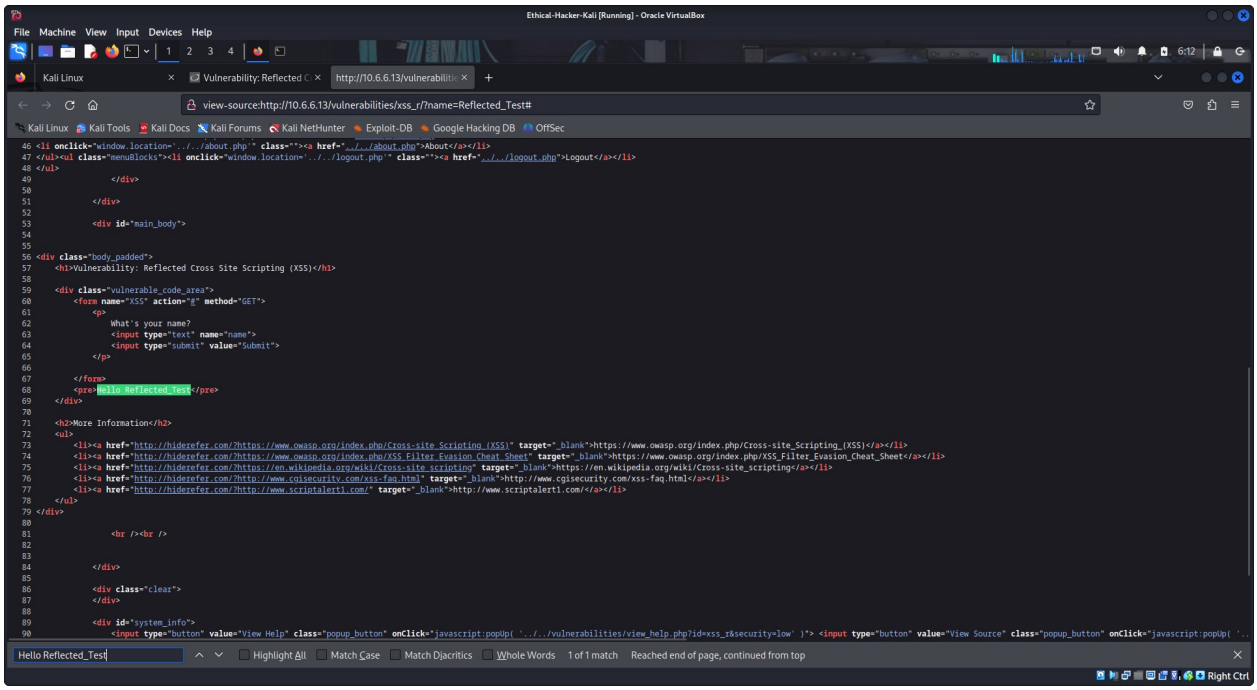
c. Type the string **Reflected_Test** in the **What's your name?** box and click **Submit**.

You will see the message **Hello Reflected_Test** appear.



d. Enter **CTRL+U** on the keyboard to view the source code of the page.
e. Search for the string **Hello Reflected_Test** by entering CTRL+F to open a search box.

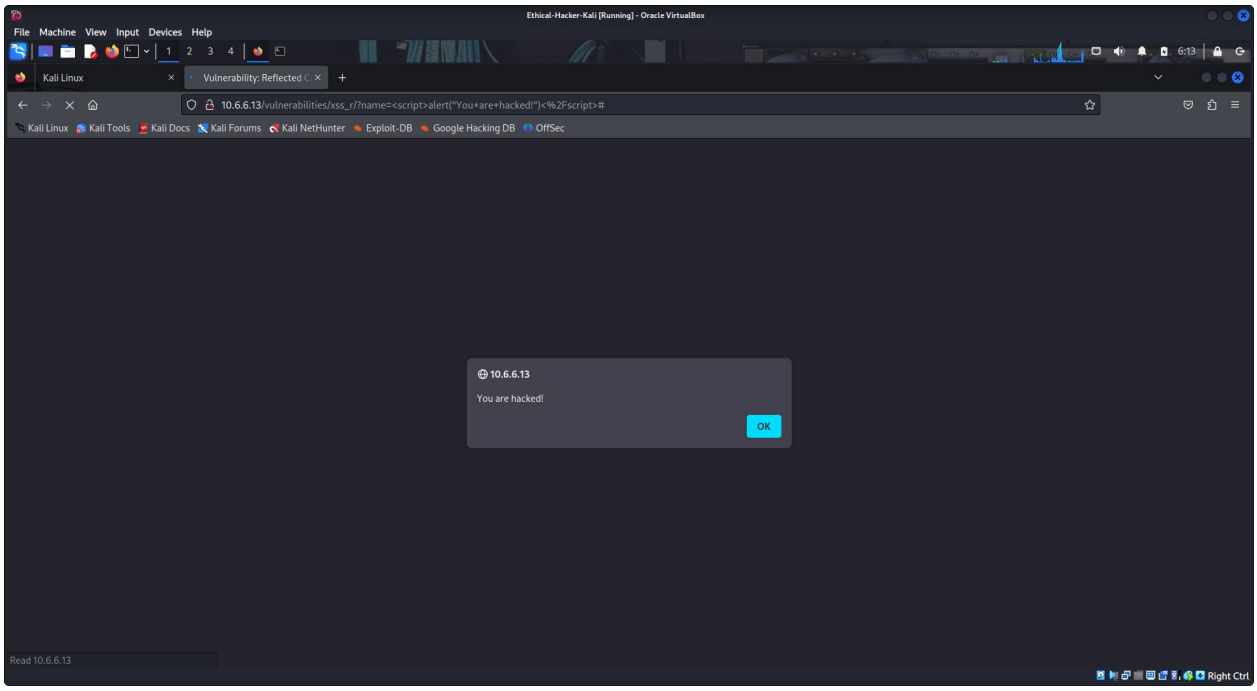
The presence of the string in the page source HTML indicates that values entered in a user response text field are inserted into the source code for the page. This indicates to an attacker that the page may be vulnerable to reflected XSS attacks.



f. Close the source code window and return to the Reflected XSS Vulnerability page.
g. Enter the following payload in the **What's your name?** box and click **Submit**.

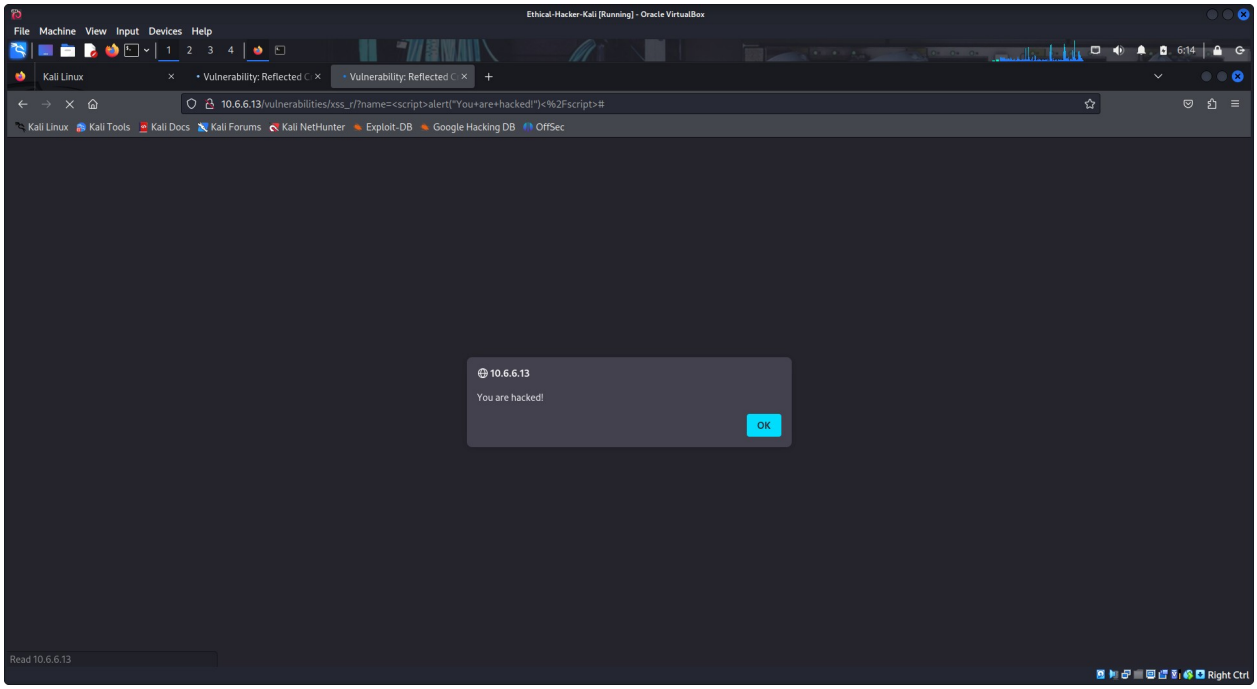
```
<script>alert("You are hacked!")</script>
```

An alert popup box will appear with the words **You are hacked!**. This means the site is vulnerable to Reflected XSS attacks and we have successfully exploited the vulnerability.



h. Select and copy the URL for the compromised page(**http://10.6.6.13/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%22You+are+hacked%21%22%29%3C%2Fscript%3E#**). Open a new browser tab and paste the URL into the URL field and press <Enter>.

You should see the same web page appear displaying the **You are hacked**! popup box. This means that if a user opens the URL a malicious script will execute. The alert box is used to simulate a malicious script in this lab.

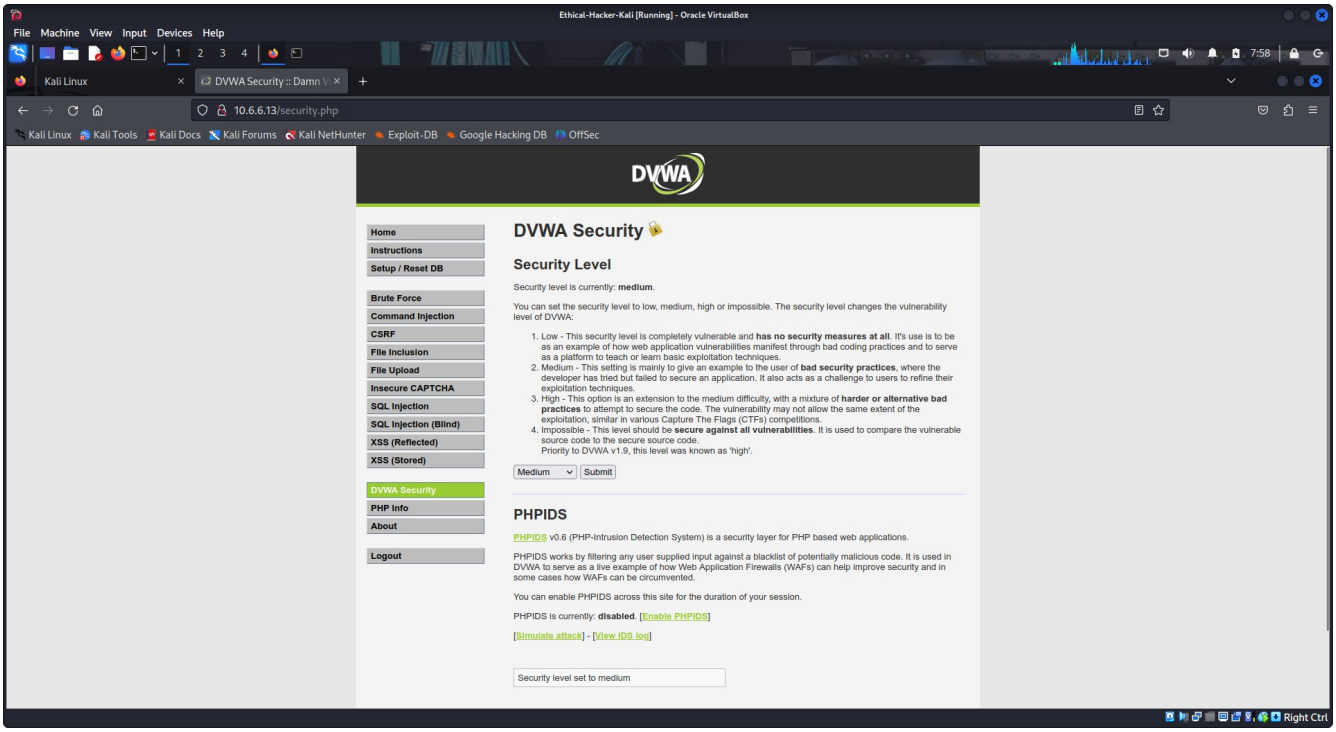


In an ethical hacking engagement, you would try inserting a simple test script into input fields to see if the script executes. If so, the website is vulnerable to reflected XSS attacks. You could then distribute the link in a phishing attack to determine the level of security awareness among your customers' employees.

Step 3: Perform a Reflected XSS attack at Medium security level.

You will attempt the same attack, but this time the security level of the Web site will Medium.

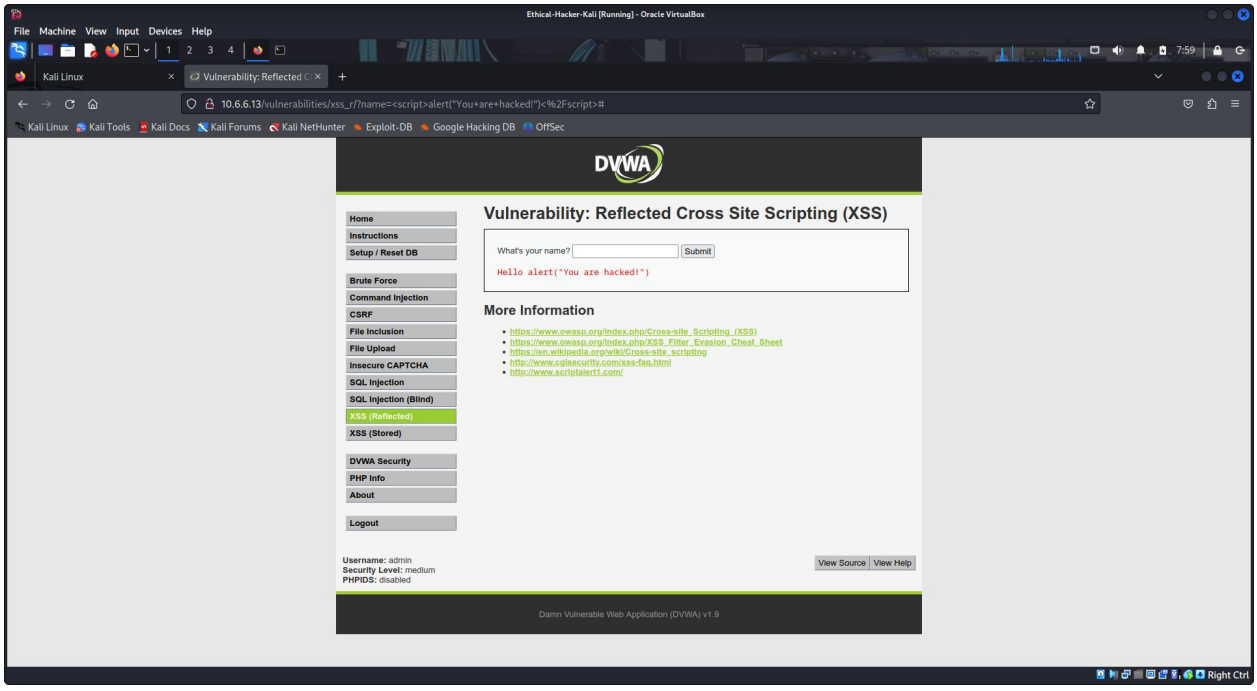
a. Select **DVWA Security** in the left menu and select **Medium** in the Security Level dropdown. Click **Submit**.



b. Select **XSS (Reflected)** in the left menu.
c. Again, enter the following payload in the **What's your name?** box and click **Submit**.

```
<script>alert("You are hacked!")</script>
```

You will see a Hello response, but this time no pop up will appear. This indicates that the script did not execute. Note that the script is displayed as literal text.



We can analyze the code in the backend of the web site to investigate the reason.

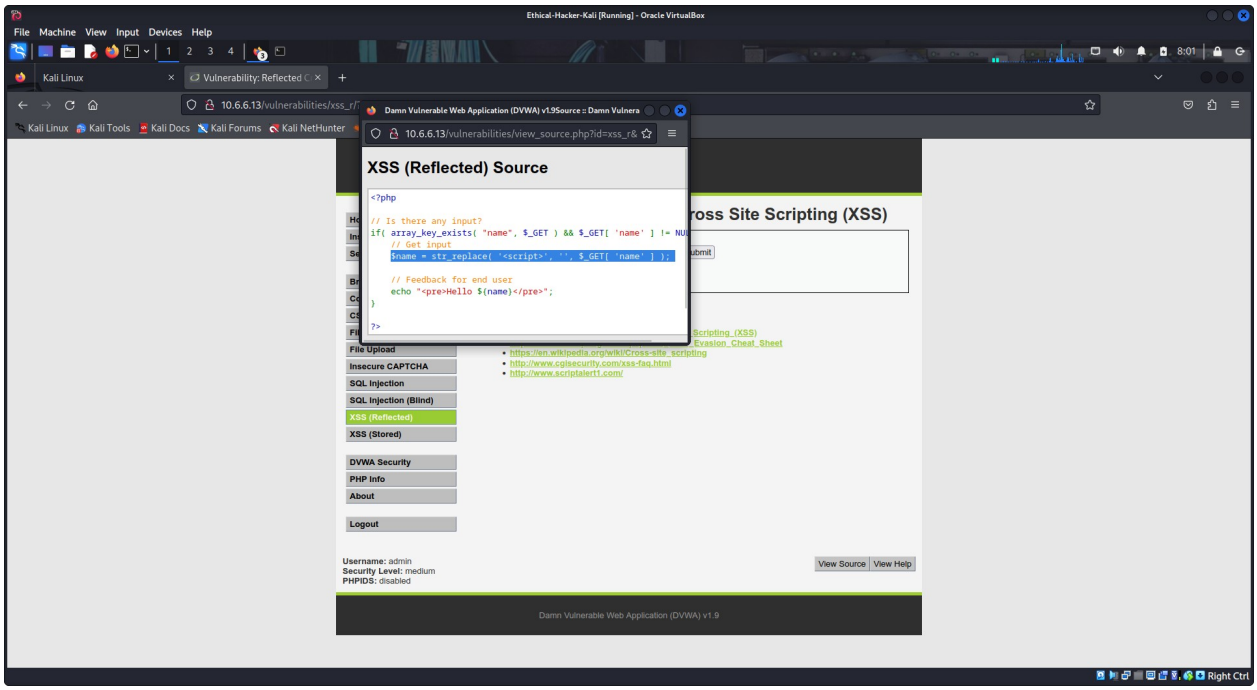
d. Click the **View Source** button on the bottom right of the page and review the PHP code.

Note: On a real web server, we would not have access to this backend source code, but here on DVWS we do.

e. Note the line:

```
$name = str_replace ( '<script>', '',  $_GET[ 'name' ] );
```

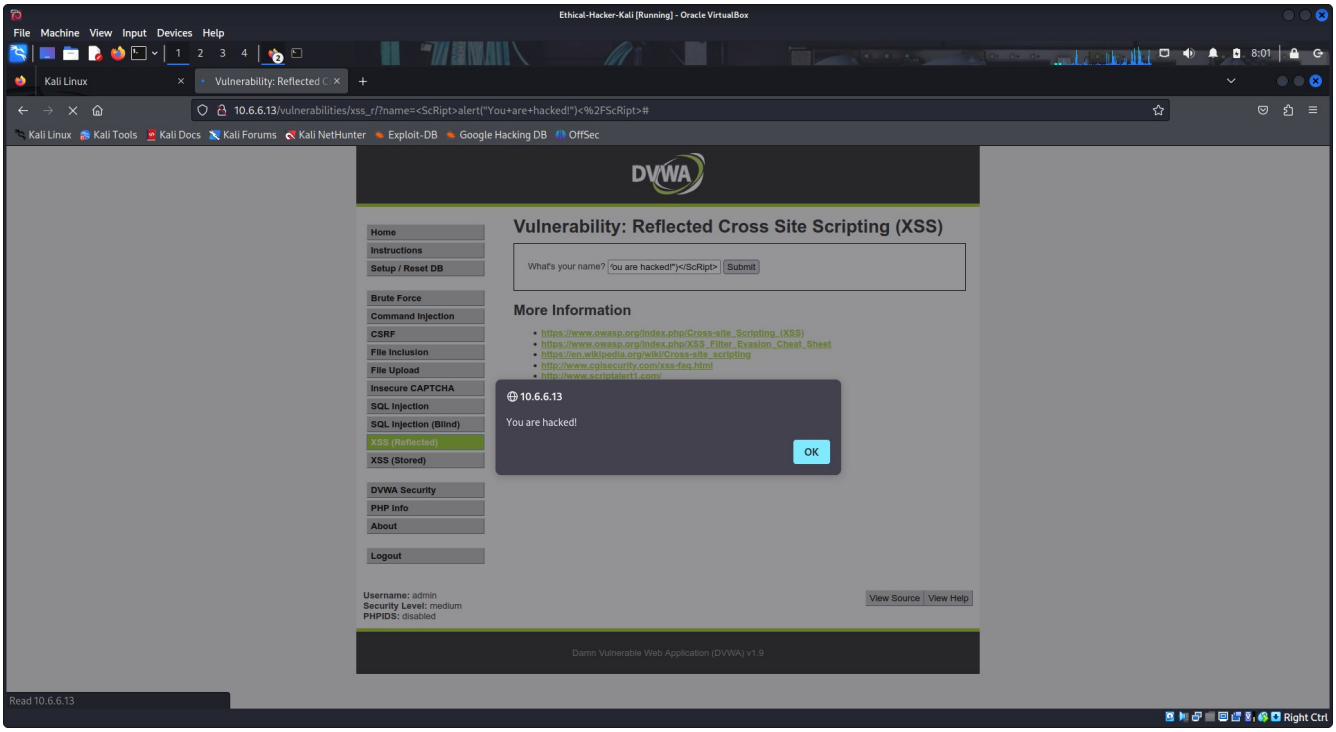
This source code creates a filter, with **str_replace()** function, that removes the **<script>** tag in our payload and replaces it with a null value. This renders the payload script ineffective, so the attack failed, and no popup window is displayed. Because this script is only filtering out <script> in lower case, we can try and get around the filter by using a different tag in the payload. We will use **<ScRipt>**.



f. Close the source code window and return to the Reflected XSS Vulnerability page.

g. Enter the following payload in the **What's your name?** box and click **Submit**.

```
<ScRipt>alert("You are hacked!")</ScRipt>
```

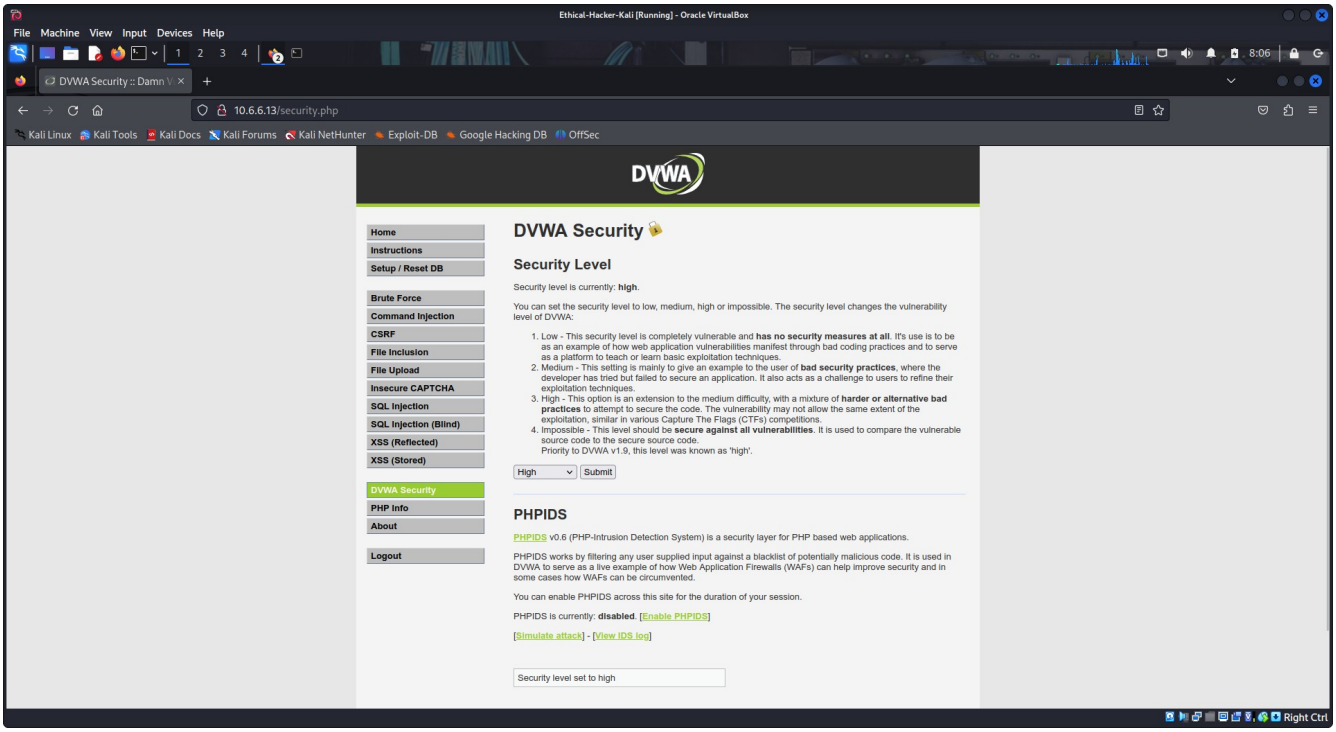


Did the popup alert appear? If so, why? **Yes, The popup appeared because the payload with the <ScRipt> tag was able to bypass the filter. This means the site is still vulnerable to Reflected XSS attacks even at the Medium security level.**

Step 4: Perform a Reflective XSS attack at High security level.

The same attack will be attempted, but this time the security level of the website will be High.

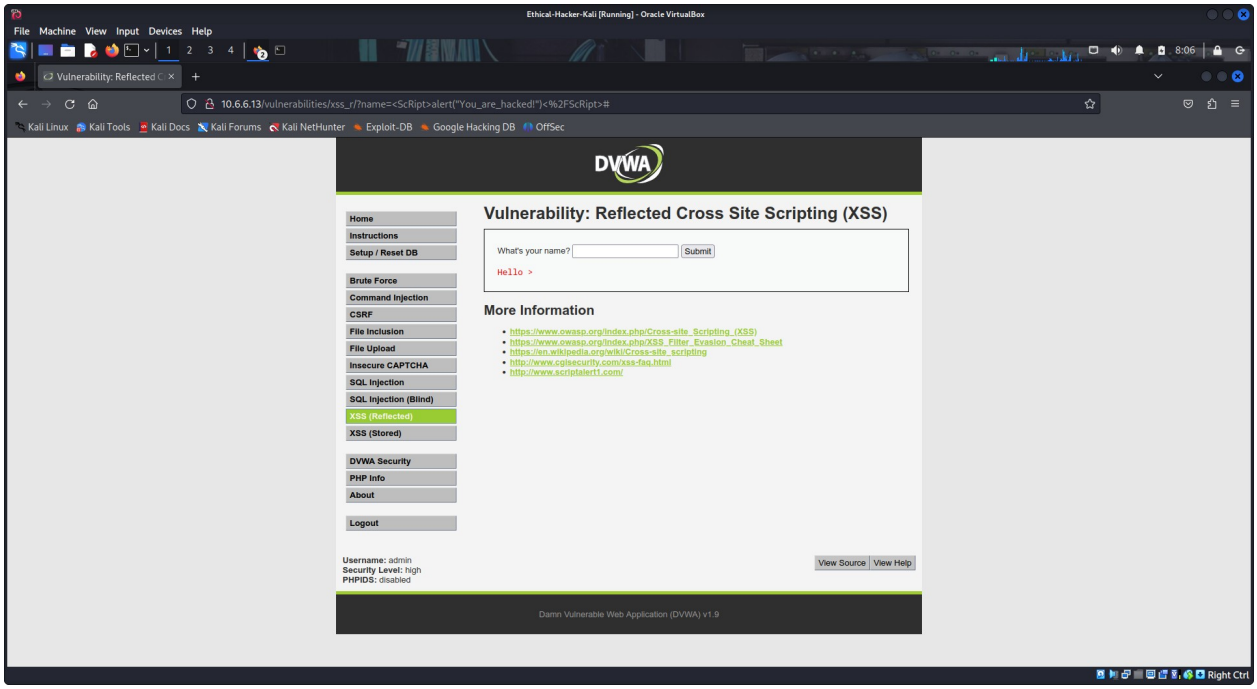
a. Select **DVWA Security** in the left menu and select **High** in the Security Level dropdown. Click **Submit**.



b. Select **XSS (Reflected)** in the left menu.
c. Enter the following payload in the **What's your name?** box and click **Submit**. (Note the use of underscores to replace spaces.)

```
<ScRipt>alert("You_are_hacked!")</ScRipt>
```

There is a Hello message and no alert pop up box. Again, we can analyze the backend source code to investigate.

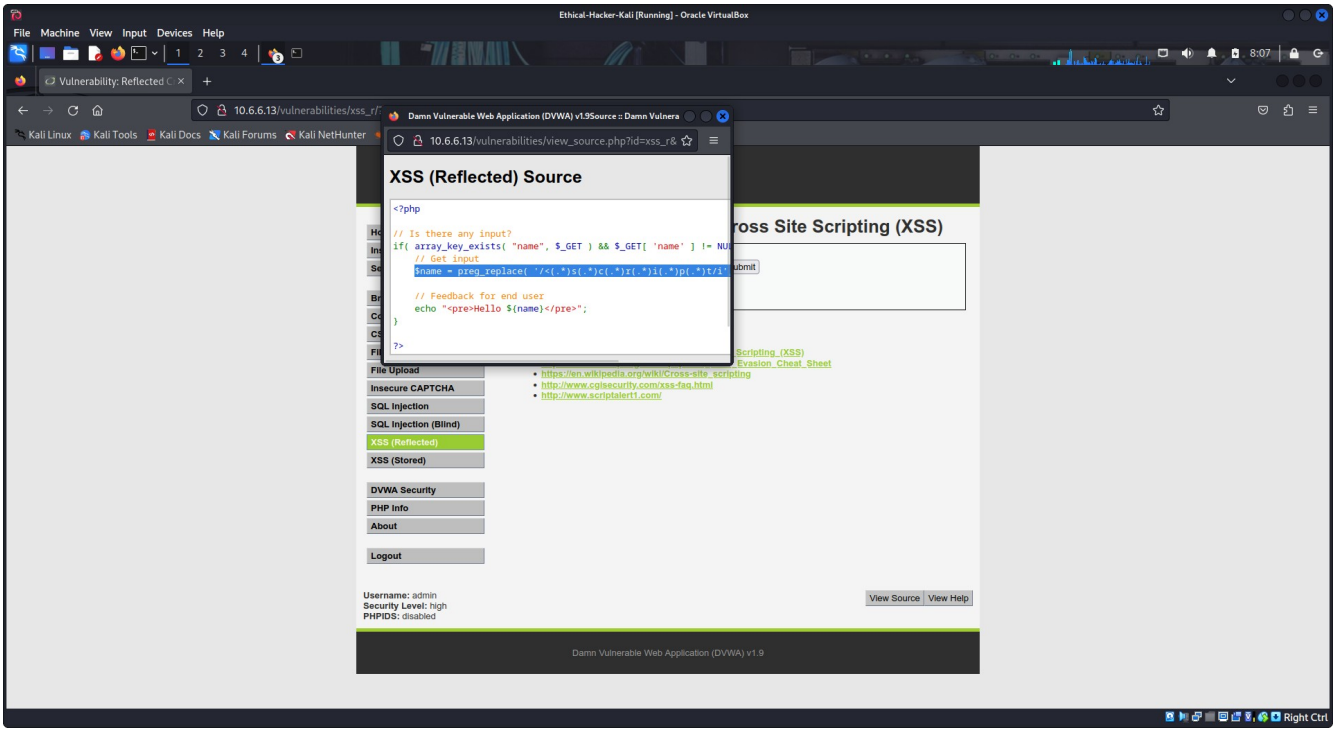


d. Click the **View Source** button and review the PHP code.

Note the following line:

```
$name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '',
$_GET[ 'name' ] );
```

In this code, the developer used a regular expression to replace any form of the **<script>** tag, no matter what case of the characters is used, with a null value.



Which character in the script was omitted from the regular expression? How do you know? **The “>” character was omitted. You can still see it in the output for the submitted “What’s your name?” input.**

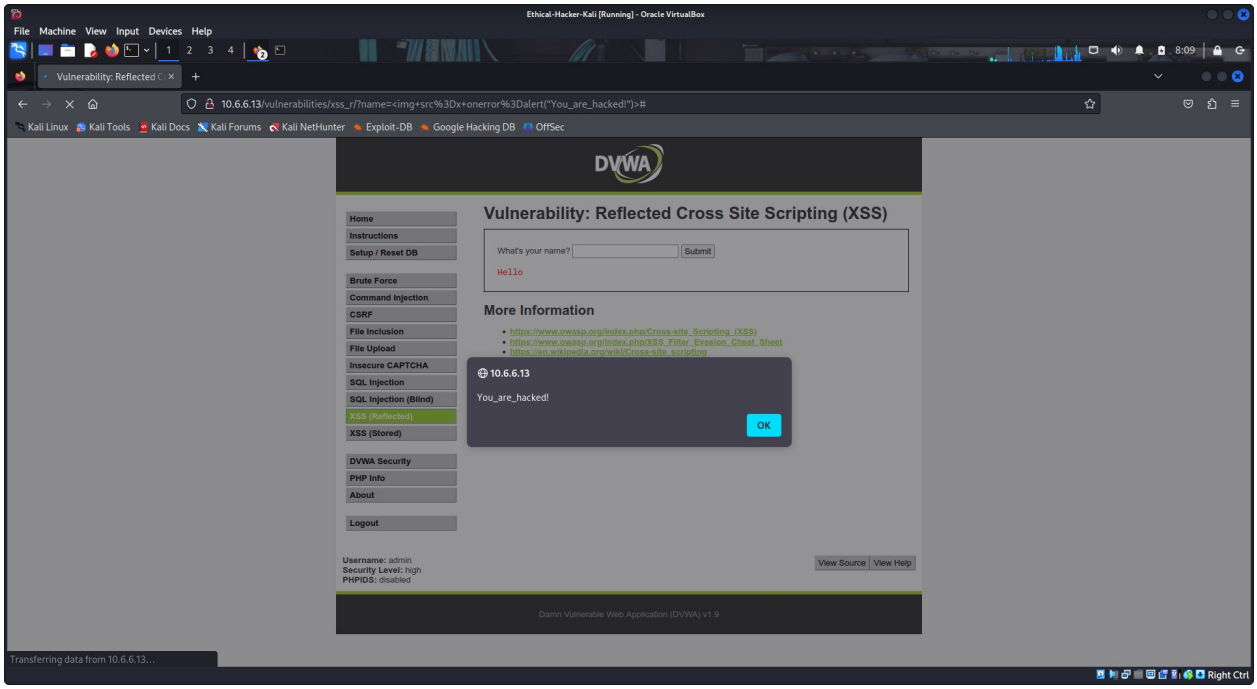
To bypass this filter, we must use another HTML tag instead of **<script>** to attack the site.

Close the source code window and return to the Reflected XSS Vulnerability page.

f. Enter the following payload in the **What's your name?** box and click **Submit**. (Note the use of underscores to replace spaces.)

<img src=x onerror=alert("You_are_hacked!")>

The XSS popup box will appear this time. We successfully bypassed the filter and exploited the Reflected XSS vulnerability in DVWA at High level security.



Review the text that you input into the web form. How did it work?
It forced an error to occur by attempting to load a non-existent image. The error was detected with onerror and the alert response was triggered to display the alert box.

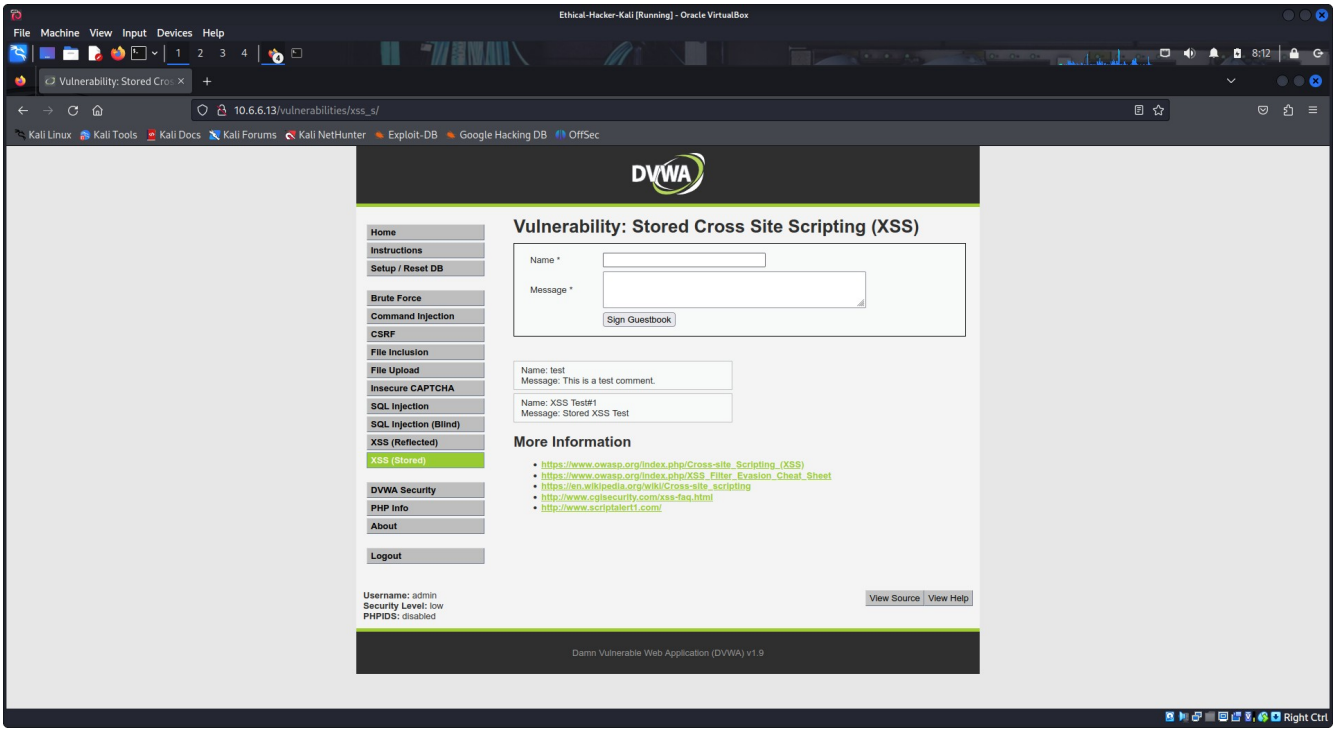
Part 2: Perform Stored Cross Site Scripting Exploits

With the stored XSS exploit, you enter a malicious script through user input and the script is stored on the target server in a message forum, database, visitor log, or comment field. When a user visits the target, the server exposes the user to the malicious code.

Step 1: Perform a Stored XSS attack at Low security level.

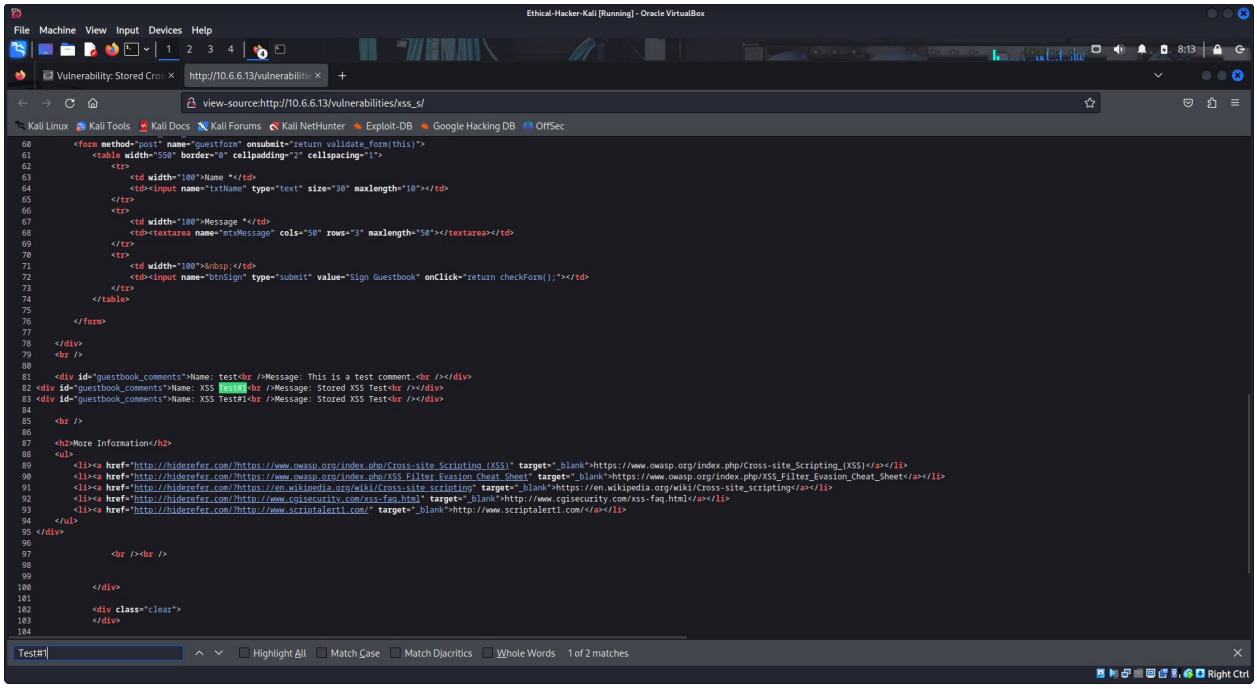
Exploiting stored XSS at low level security is easy because there are no security measures in place. You can simply submit a <script> to achieve the exploit.

a. Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**.
b. Select **XSS (Stored)** in the left menu.
c. Type the string **XSS Test#1** in the **Name*** field and type **Stored XSS Test** in the **Message *** field. click **Sign Guestbook**. You will the result below



d. Enter **CTRL+U** on the keyboard to view the page source code. Enter **CTRL+F** to search for the **Test#1** and **Stored XSS Test** strings.

Both strings, **Test#1** and **Stored XSS Test**, will be in the page source code indicating that the two input fields may be vulnerable to a Stored XSS attack.

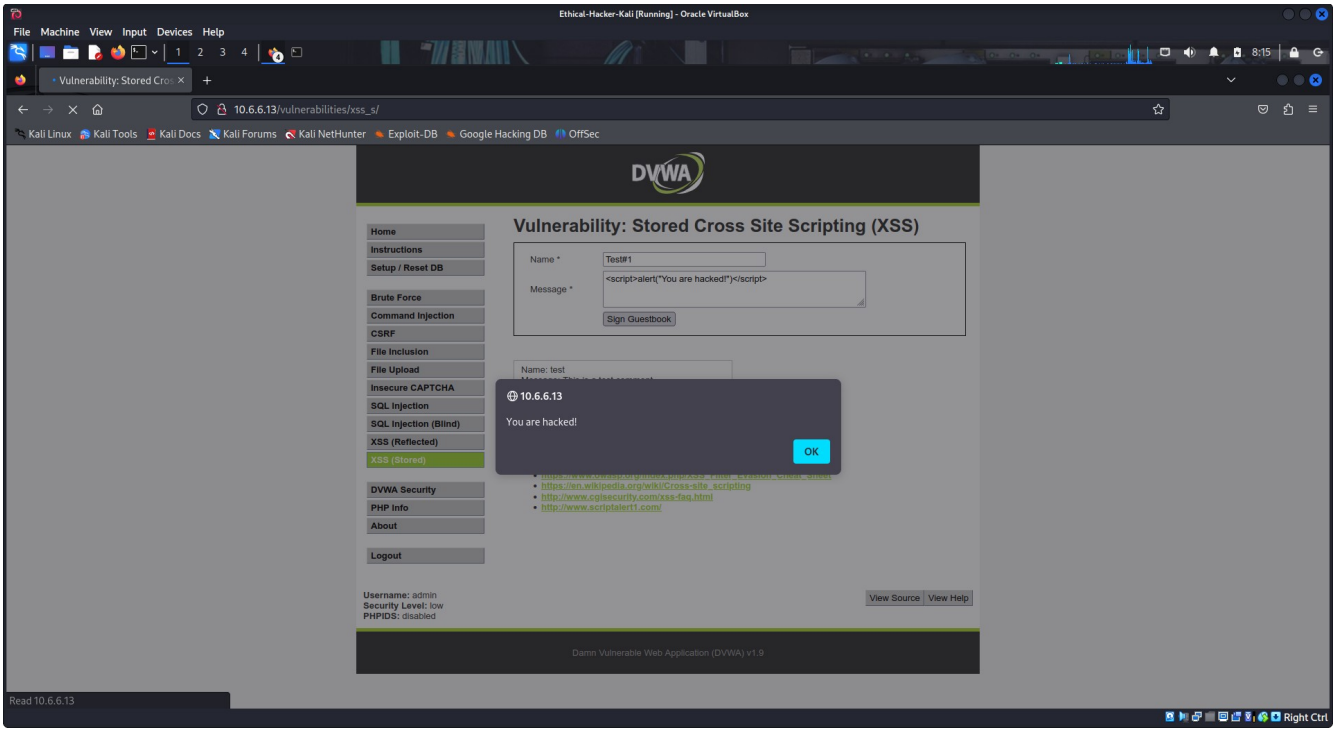


e. Close the source code window and return to the Stored XSS Vulnerability page.

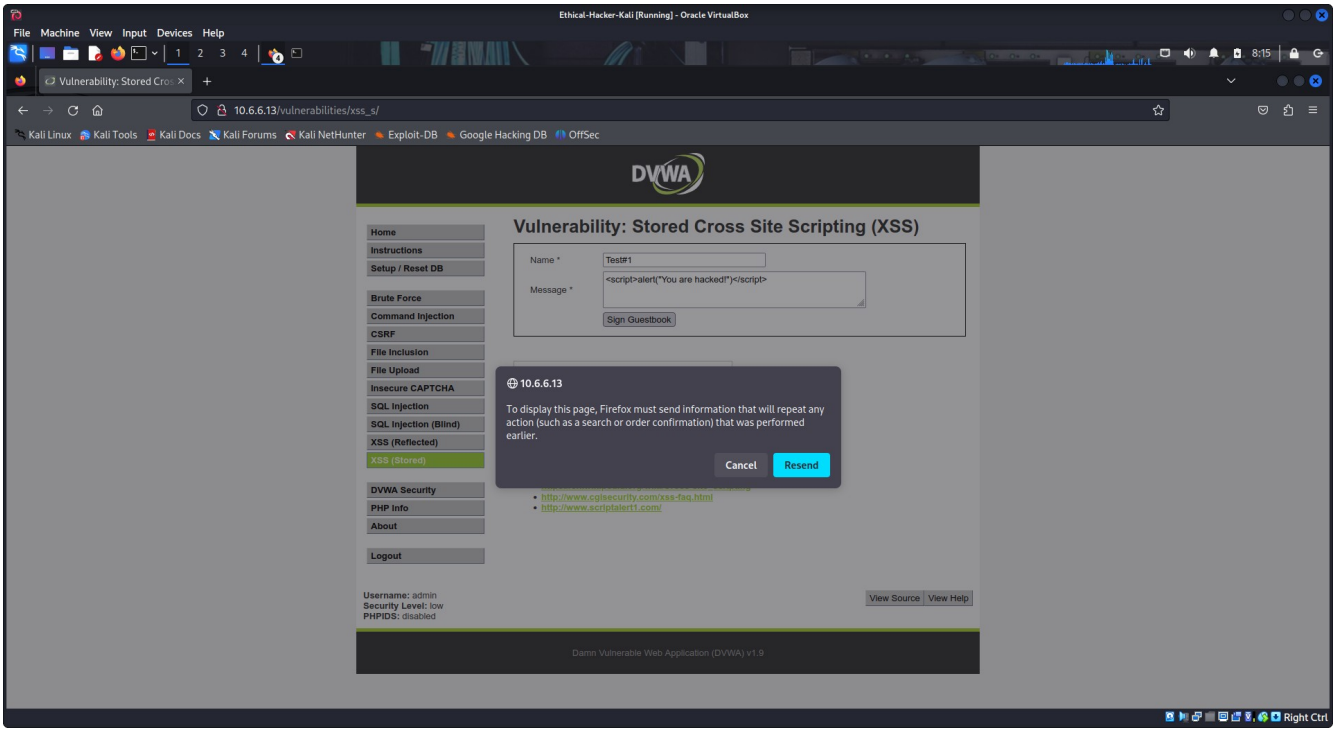
f. Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** field and click **Sign Guestbook**.

```
<script>alert("You are hacked!")</script>
```

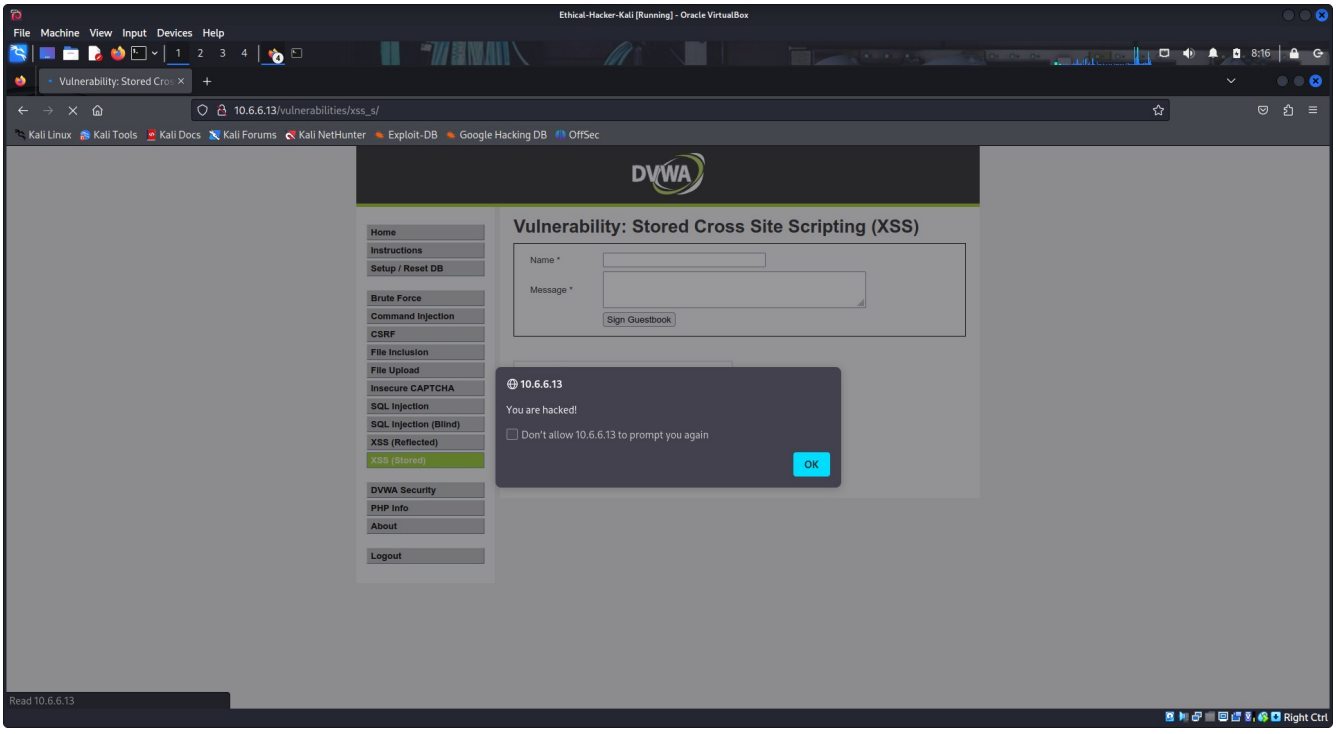
An XSS alert popup box will appear with the words **You are hacked!**. This means the site was vulnerable to stored XSS attacks and we have successfully exploited the vulnerability.



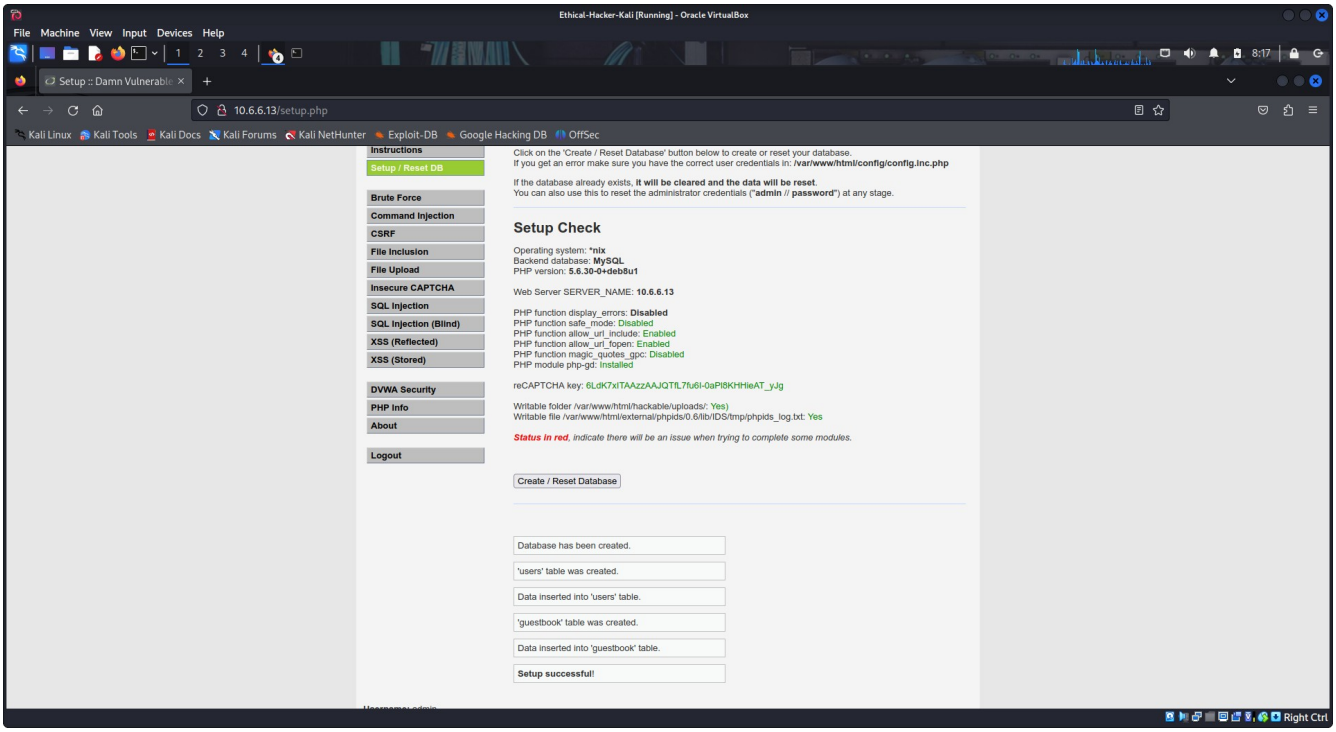
g. Refresh the page. If alerted, click **Resend** to display the page. The XSS alert popup box will appear again.



Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed.



h. Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**.

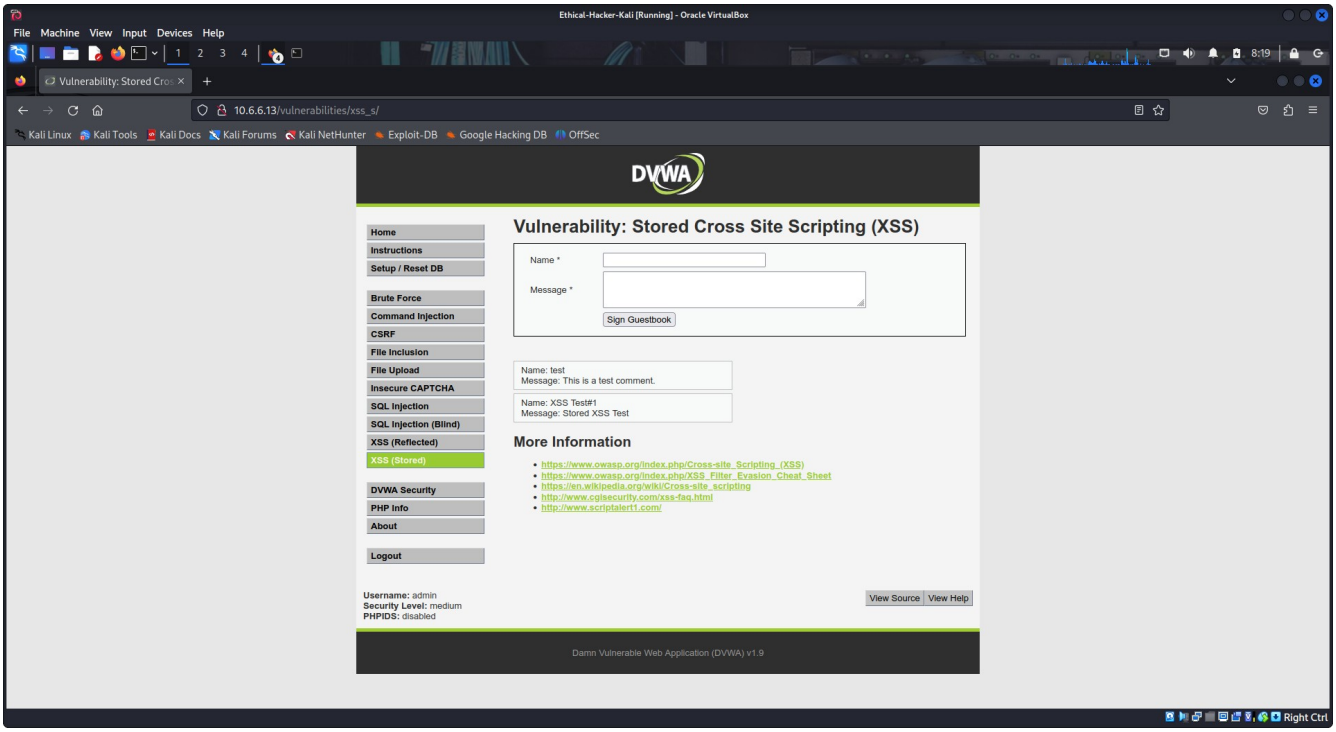


Step 2: Perform a Stored XSS attack at Medium security level.

The same attack will be attempted, but this time the security level of the Web site will be changed to Medium.

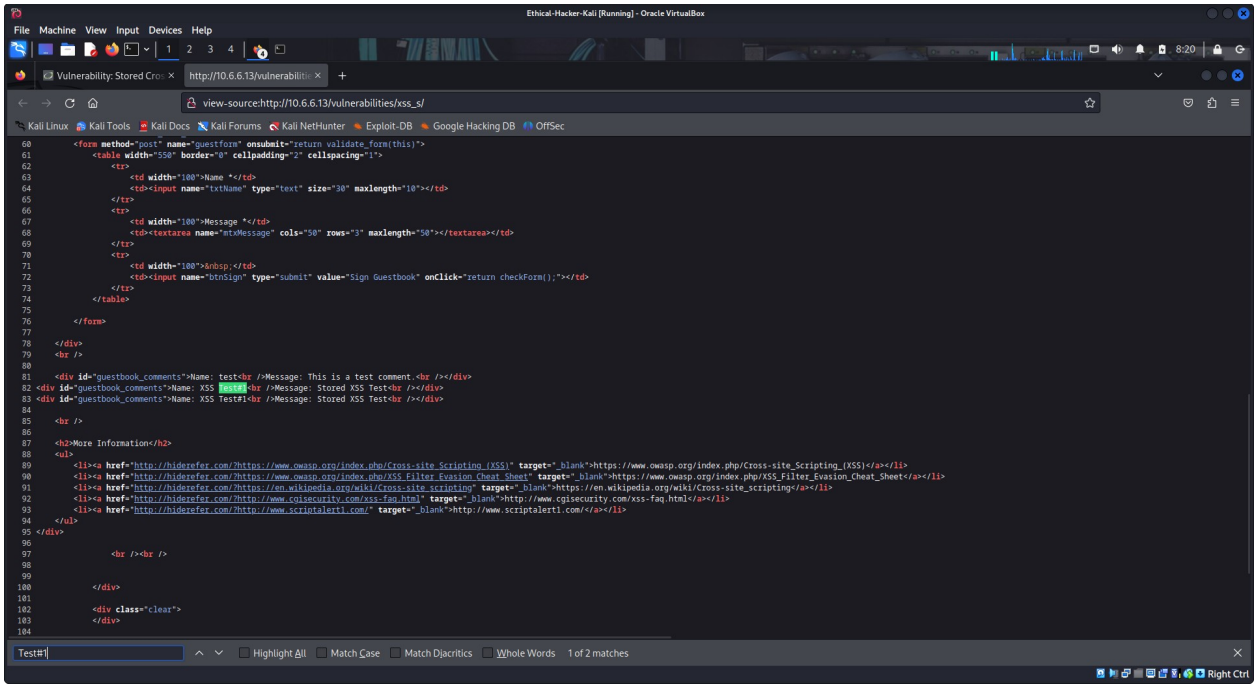
Exploiting reflected XSS at medium level security is not difficult. Using <script> will be rejected but changing it to a different case such as <ScRipt> will bypass the security and achieve the exploit.

a. Select **DVWA Security** in the left menu and select **Medium** in the Security Level options dropdown. Click **Submit**.
b. Select **XSS (Stored)** in the left menu.
c. Type the string **XSS Test#1** in the **Name*** field and type **Stored XSS Test** in the **Message *** field. click **Sign Guestbook**.



d. Enter **CTRL+U** on the keyboard to view the page source code. Enter **CTRL+F** to search for the **Test#1** and **Stored XSS Test** strings.

Both strings will be in the page source code indicating that the two input fields may be vulnerable to a Stored XSS attack.



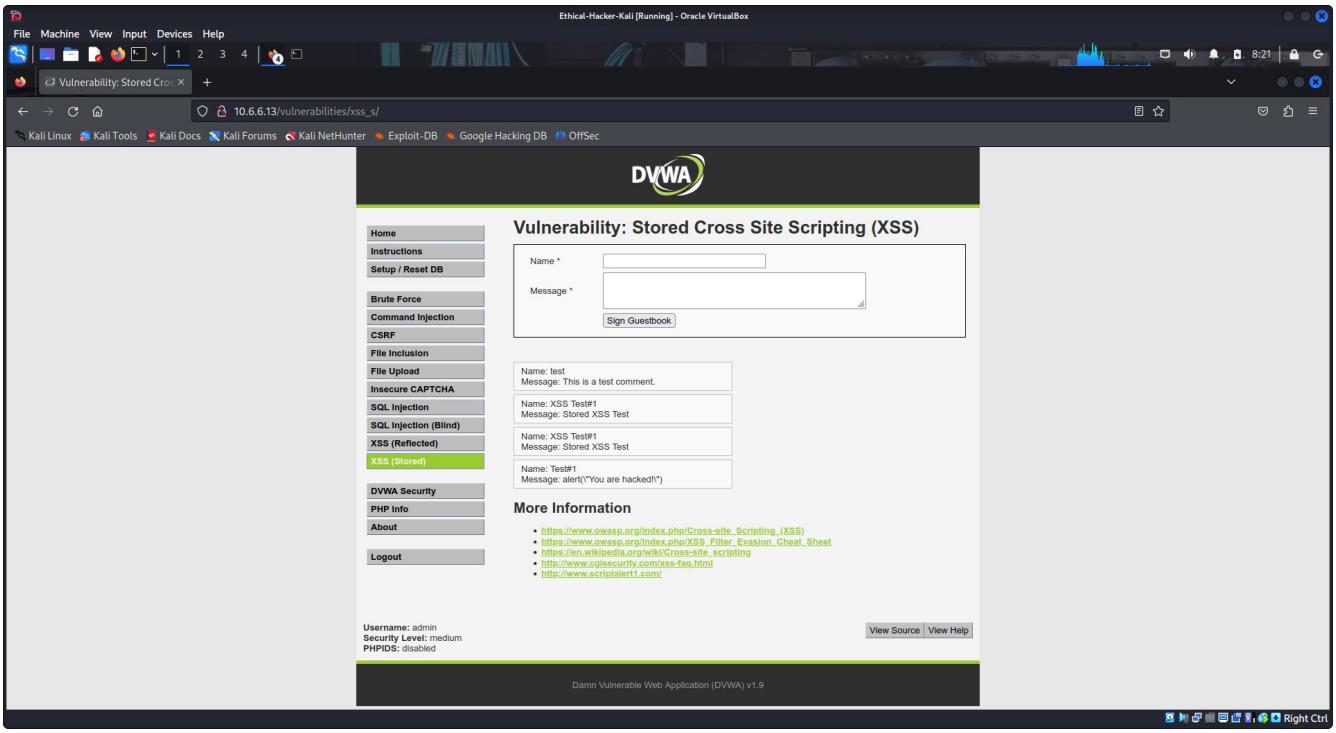
e. Close the source code window and return to the Vulnerability page.

f. Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** box and click **Sign Guestbook**.

```
<script>alert("You are hacked!")</script>
```

No popup box should appear. Refreshing the page should not cause the alert popup box to appear either.

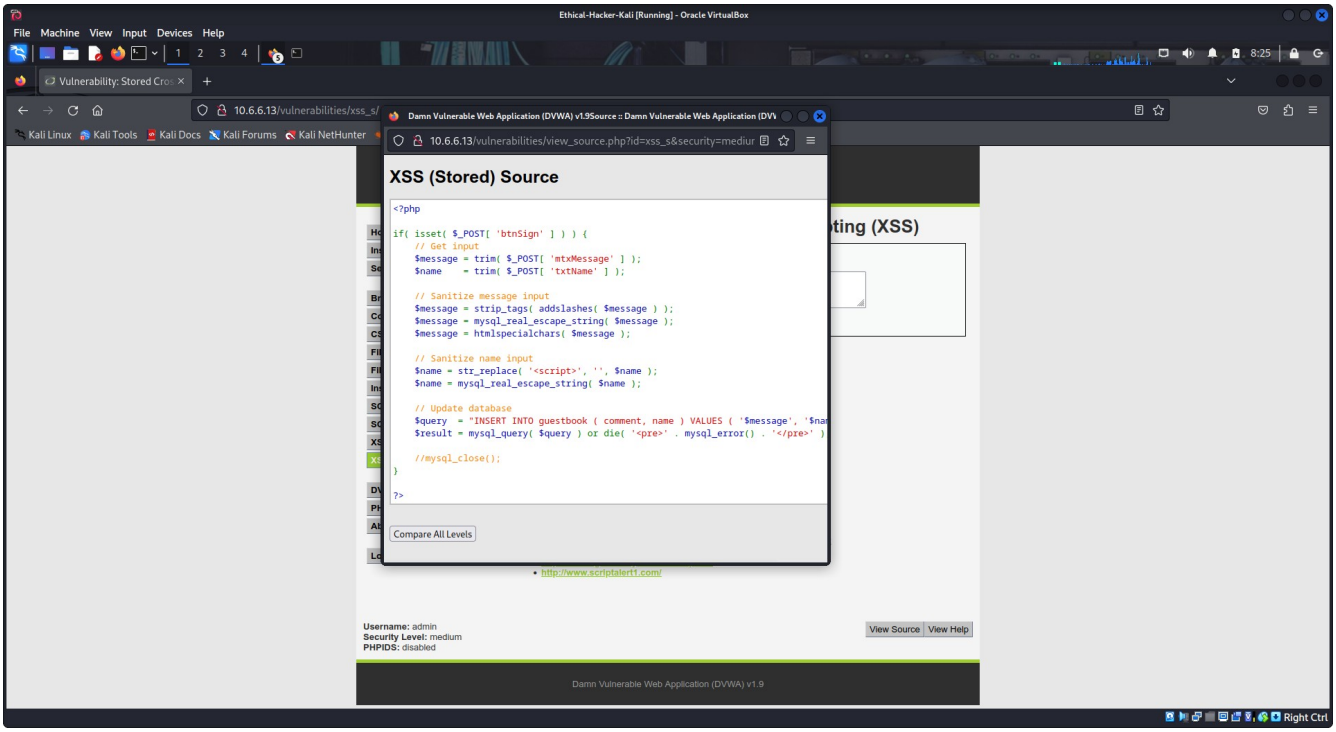
This means that there is code in the backend that is sanitizing the user input from the **Message *** field to prevent scripts from being submitted. You can see the modified input in the last rectangle message box below the input fields.



How did the input filter script modify the input? **It removed the html <script> tags and added slashes.**

- Click the **View Source** button and review the PHP source code and investigate.

You will see two blocks of code with the word **Sanitize**. The first block of code, under **// Sanitize message input**, contains two PHP functions for performing input sanitization. The **strip_tags()** function removes all html tags from the message field before storing them in the database. The **htmlspecialchars()** function converts all special characters into equivalent HTML entities so they are not reflected back in the browser.

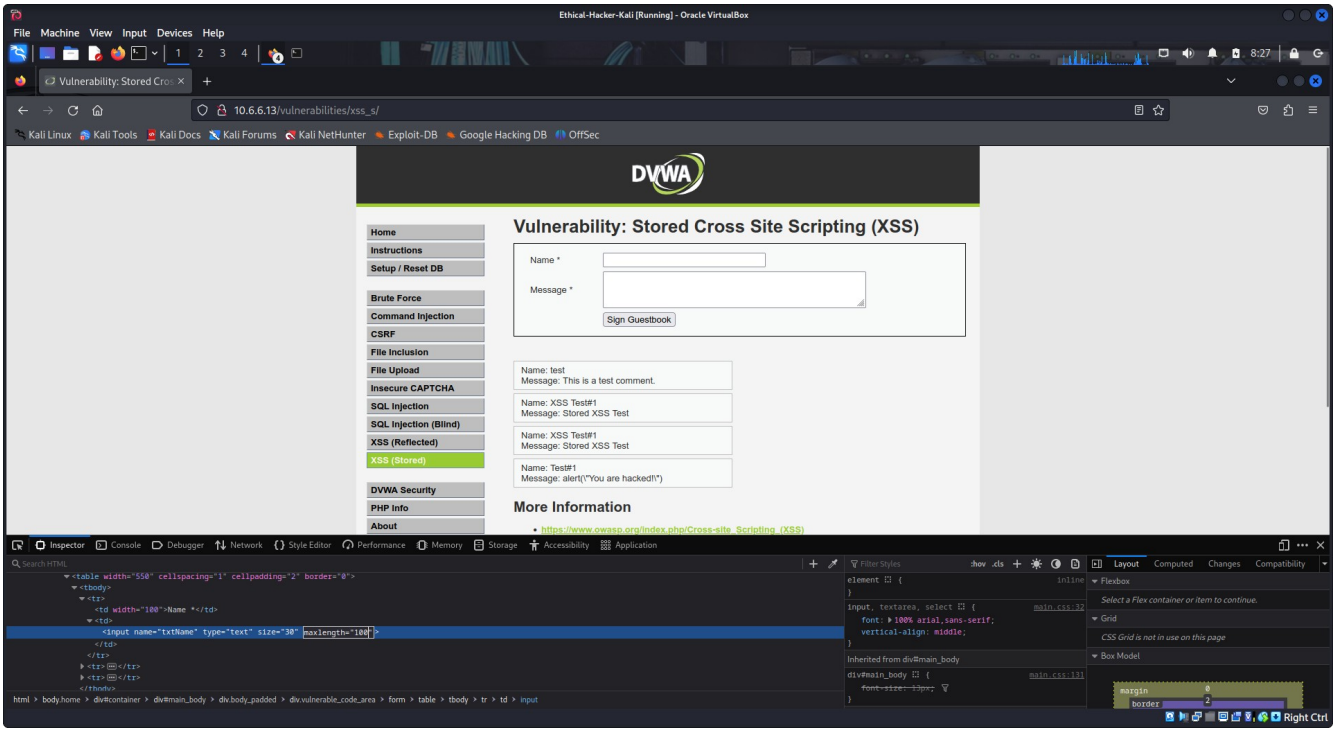


Research the PHP addslashes() function on the internet. How did it change the input? **It added slashes before the quotation marks in the alert text. This "escaped" the quotation mark characters and which made them display in the output.**

The second block of code, under **// Sanitize name input**, performs input sanitation on the **Name *** field. It contains the **str_replace()** function which replaces any occurrence of the **<script>** tag with a null value. This disables the script completely.

> We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags.

h. Close the source code window.
i. Before entering any payload into the **Name *** field, the max character length restriction of 10 characters on the field must be increased. This is a client-side setting so it is easy to change with the following steps:
    1. Right-click in the **Name *** field and select **Inspect**. This opens the Web Developer Tools window and displays the page source code.
    2. Find and double-click **maxlength** in the page source and change it from **10** to **100**. The maxlength property is inside the <input> tag for the text field.



3. Press **Enter** on the keyboard to apply the changes.
4. Close the Web Developer's Tools Window.

With the **maxlength** restriction changed, the XSS payload can now be entered into the **Name *** field.

Note: Changing the **maxlength** parameter does not persist. If you refresh the page, for example, the setting needs to be changed again.

j. Return to the Vulnerability page and enter the following payload in the **Name *** field.

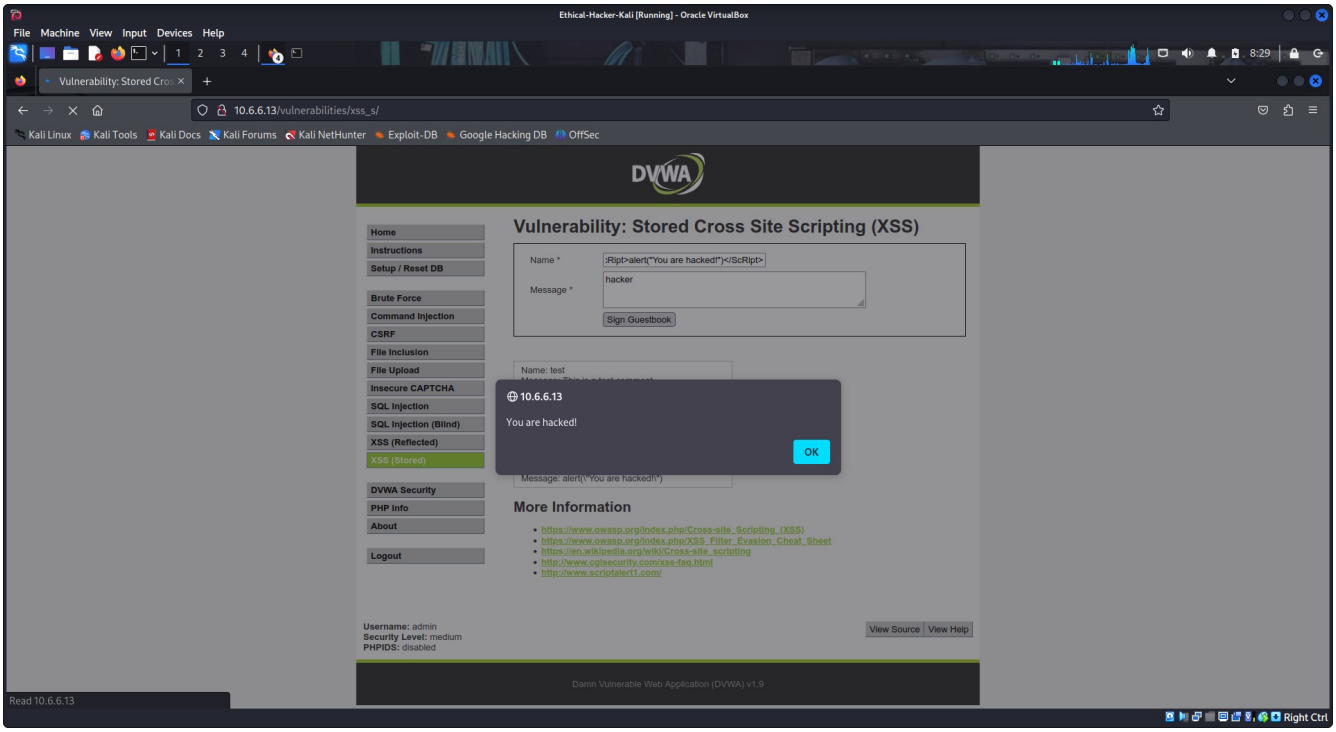
```
<ScRipt>alert("You are hacked!")</ScRipt>
```

k. In the **Message *** field you can type any text you like and then click **Sign Guestbook**.

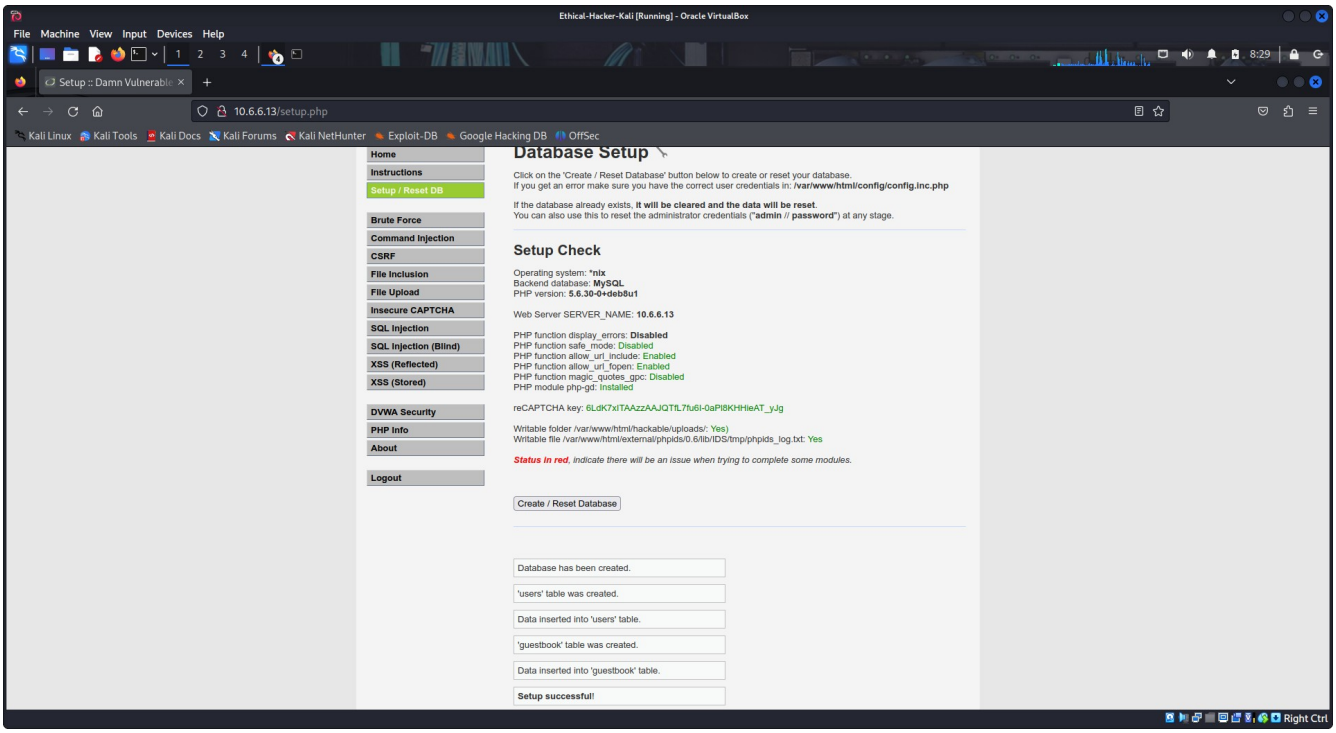
An XSS alert popup box will appear with the words **You are hacked!**.

Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed or each time other users visit the page.

The popup confirms you have successfully exploited Stored XSS vulnerability at the Medium level of security.



l. Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**.



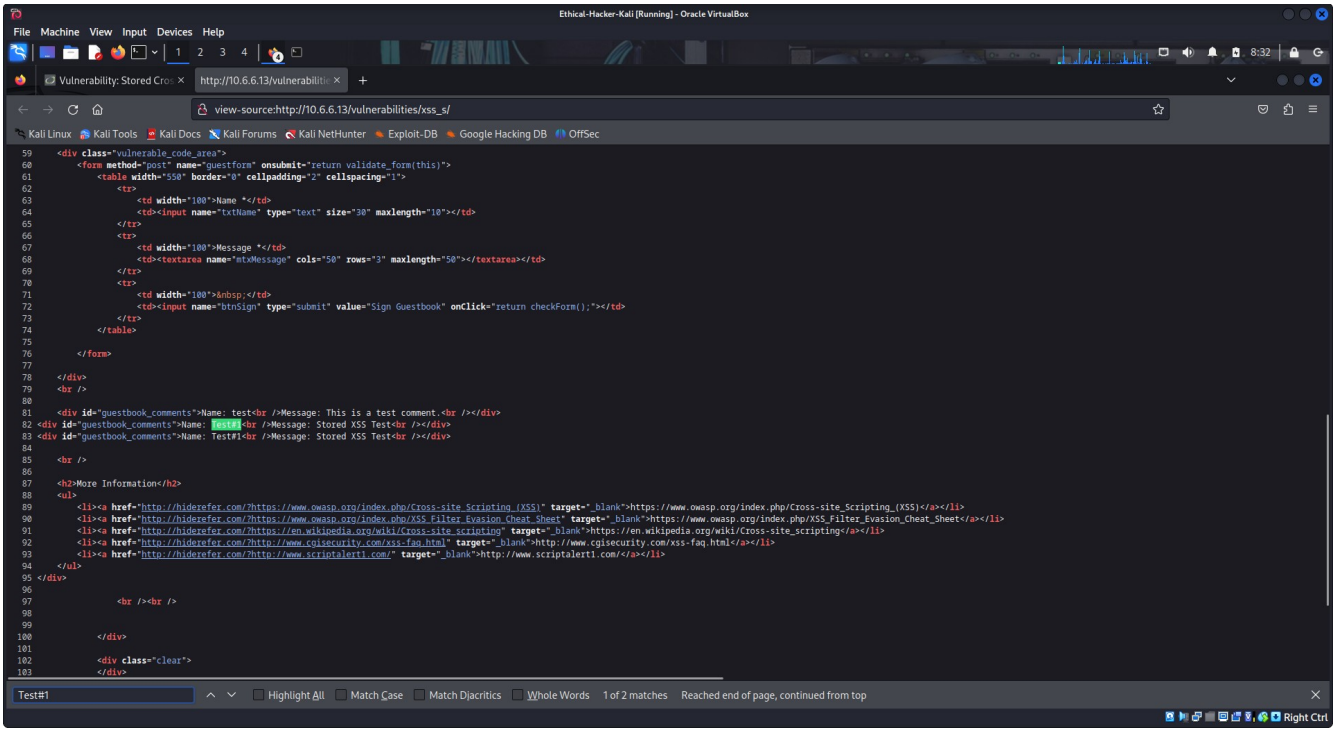
Step 3: Perform a Stored XSS attack at High security level.

You will attempt the same attack, but this time the security level of the Web site will be set to High.

At high level the security code whitelists <scripts> and all else is rejected. However, other tags, such as svg will work.

a. Select **DVWA Security** in the left menu and select **High** in the Security Level dropdown. Click **Submit**.
b. Select **XSS (Stored)** in the left menu.
c. Type the string **Test#1** in the **Name*** field and type **Stored XSS Test** in the **Message *** field. Click **Sign Guestbook**.
d. Enter **CTRL+U** on the keyboard to view the page source code. Enter **CTRL+F** to search for the **Test#1** and **XSS Test** strings.

   Both **Test#1** and **XSS Test** should be in the page source code indicating that the two input fields may be vulnerable to a Stored XSS attack.



e. Close the page source code tab and return to the Vulnerability page.
f. Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** box and click **Sign Guestbook**.

```
<ScRipt>alert("You are hacked!")</ScRipt>
```

   No popup box will appear. Refreshing the page will not cause the alert popup box to appear either.

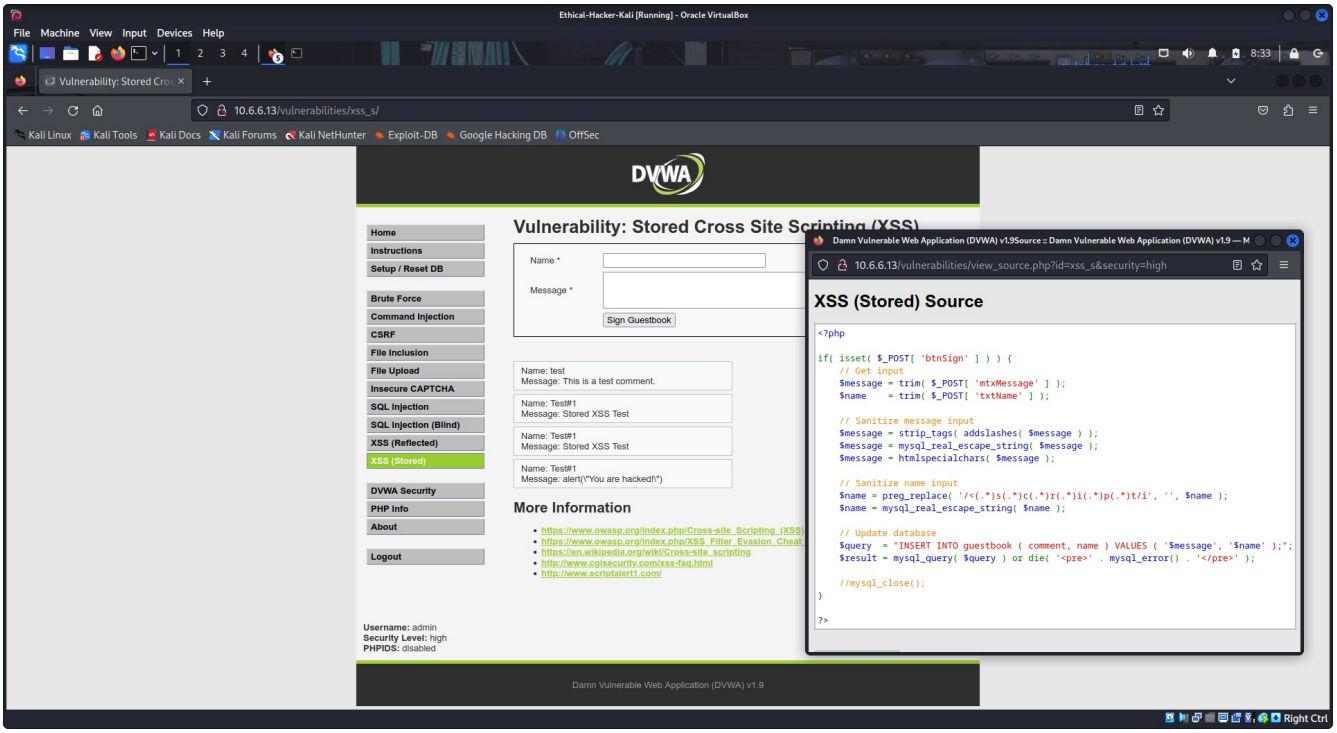
   This means that there is code in the site backend that is sanitizing the user input from the **Message *** field.

g. Click the **View Source** button and review the PHP source code and investigate.

You will see two blocks of code. As before with the Medium security, the first block of code, under // **Sanitize message input**, contains two php functions for performing input sanitization. The **strip_tags()** function removes all html tags from the message field before storing them in the database. The **htmlspecialchars()** function converts all special characters into equivalent HTML characters so they are not reflected back in the browser.

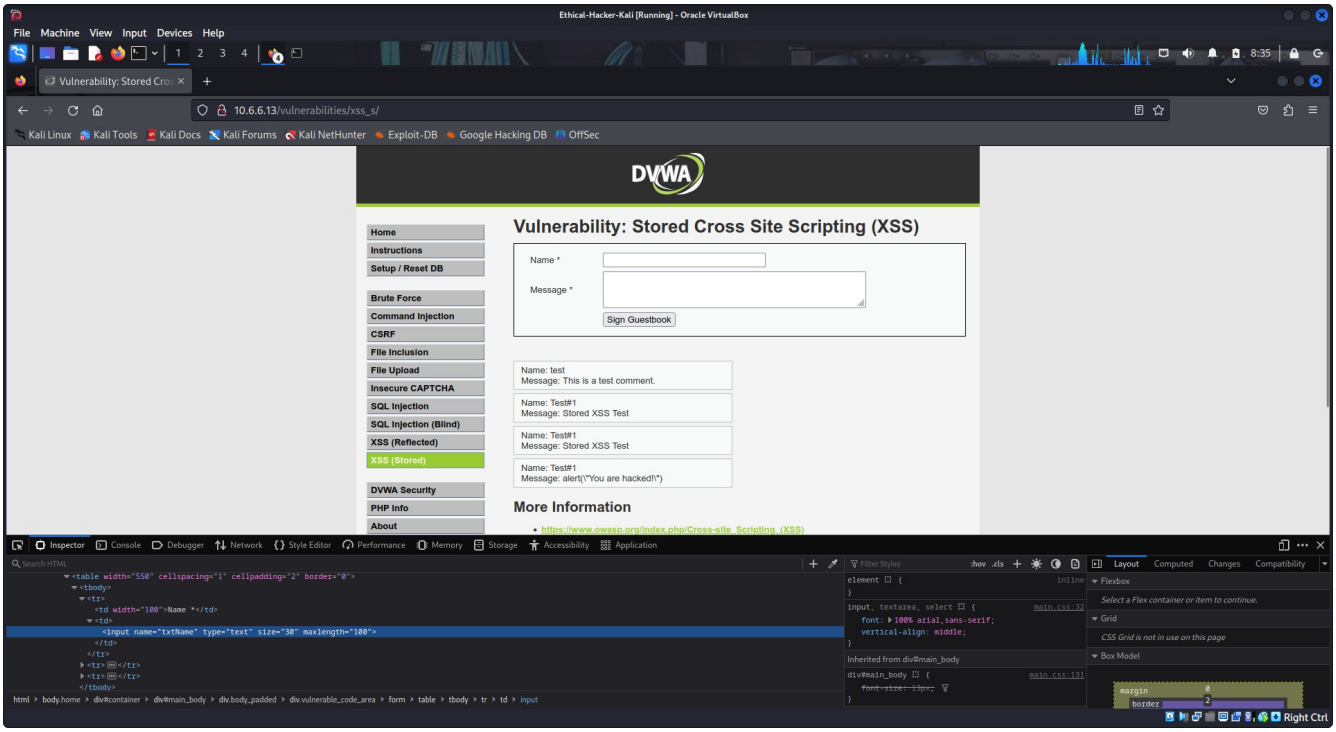
The second block of code, under // **Sanitize name input**, is performing input sanitation on the **Name *** field. It contains the **preg_replace()** function. This function uses a regular expression to replace any occurrence of the **<script>** tag, regardless of character case, with a null value.

We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags.



Before entering any payload into the **Name *** field, it will be necessary to change the max character length restriction on the field, as was done above.

With the **maxlength** restriction changed, the XSS payload can now be entered into the **Name *** field.



i. Return to the Vulnerability page and enter the following payload in the **Name *** field. (Note the use of underscores to replace spaces.)

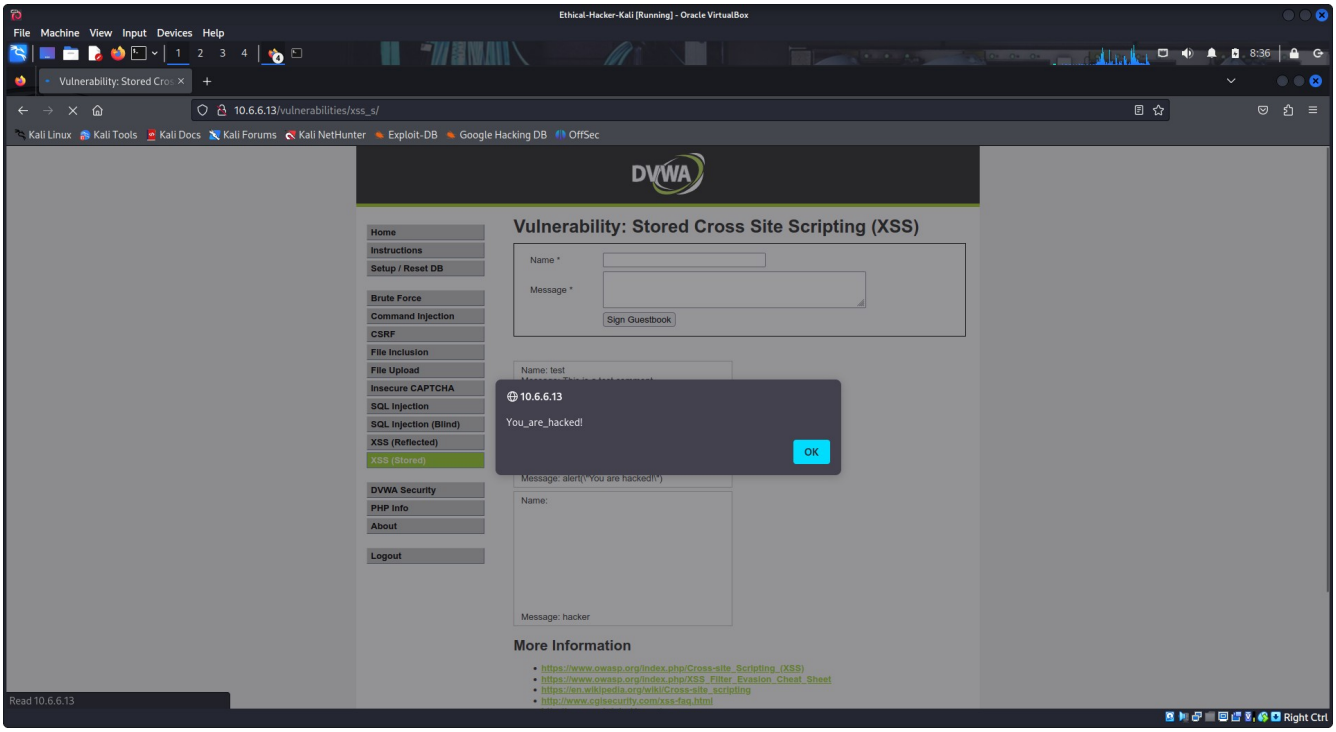
```
<svg onload=alert("You_are_hacked!")>
```

j. In the **Message *** field, you can type any text you like and then click **Sign Guestbook**.

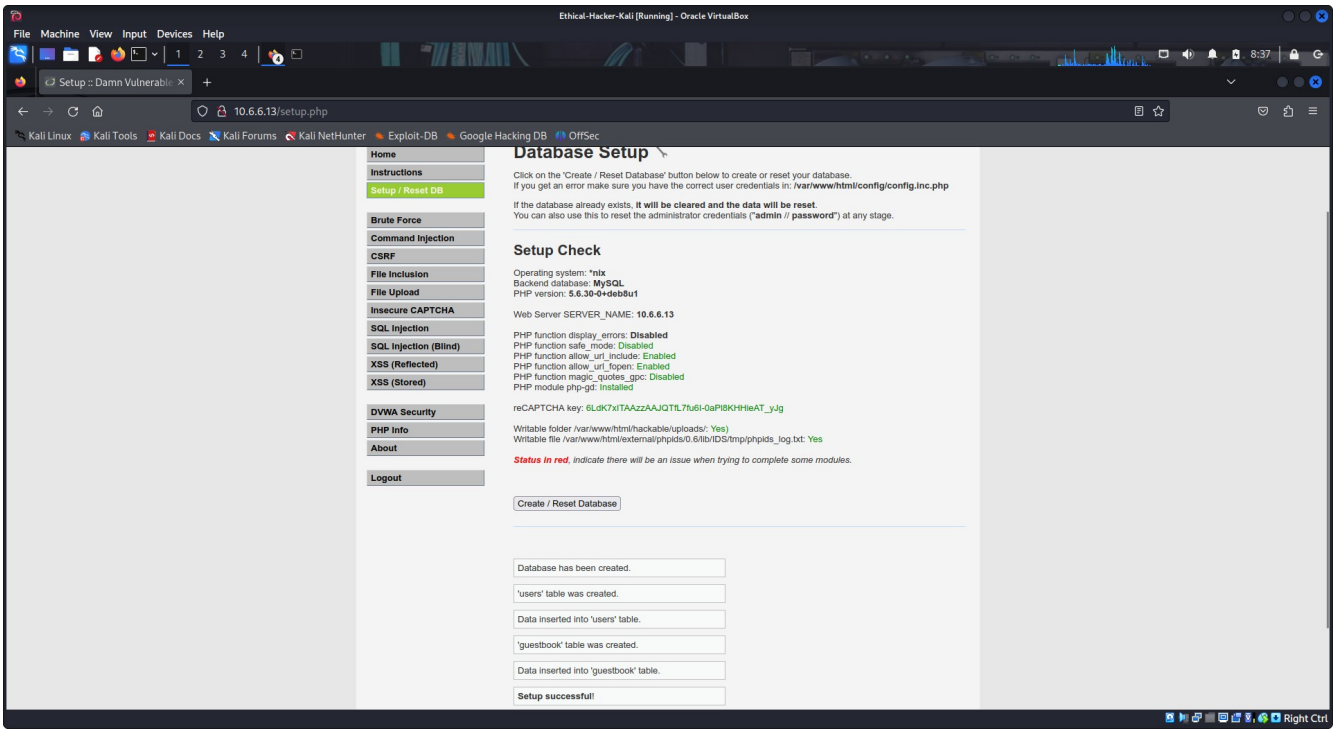
An XSS alert popup box will appear with the message "**You_are_hacked!**".

Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed.

The popup confirms you have successfully exploited a Stored XSS vulnerability at High security level.



k. Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**.



Step 4: Perform a stored iframe exploit.

a. Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**.

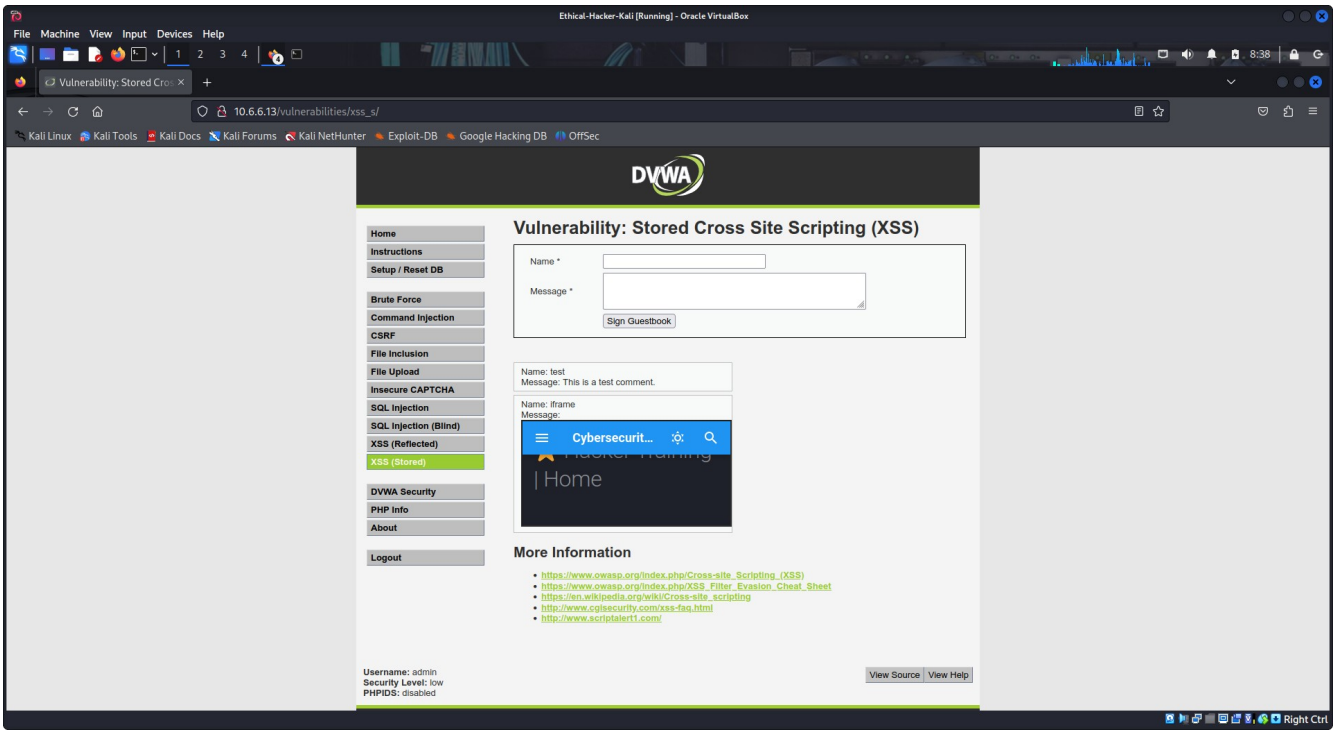
b. Select **XSS (Stored)** in the left menu.

c. Type the string **iframe** in the **Name*** field and type the following message in the **Message *** field. click **Sign Guestbook**.

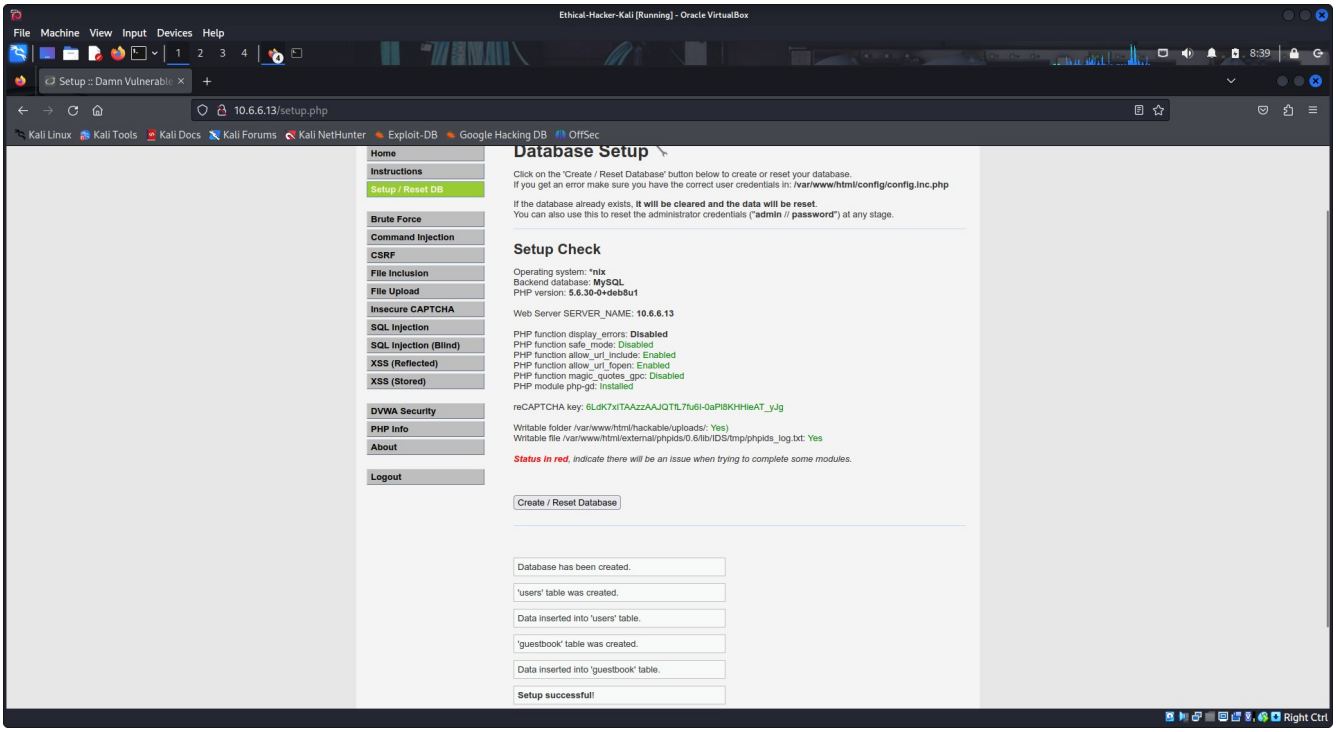
```
<iframe src="http://h4cker.org"></iframe>
```

The H4cker website should now be displayed under the iframe test message.

This is a powerful exploit because the threat actor could send the browser to a malicious website.



d. Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**.



Step 5: Perform a stored cookie exploit.

Stealing the cookies of website visitors has security implications. Cookies contain information about how and when users visit a web site and sometimes authentication information, such as usernames and passwords. Without proper security measures, a threat actor can capture cookies and use them to impersonate specific users and gain access to their information and accounts.

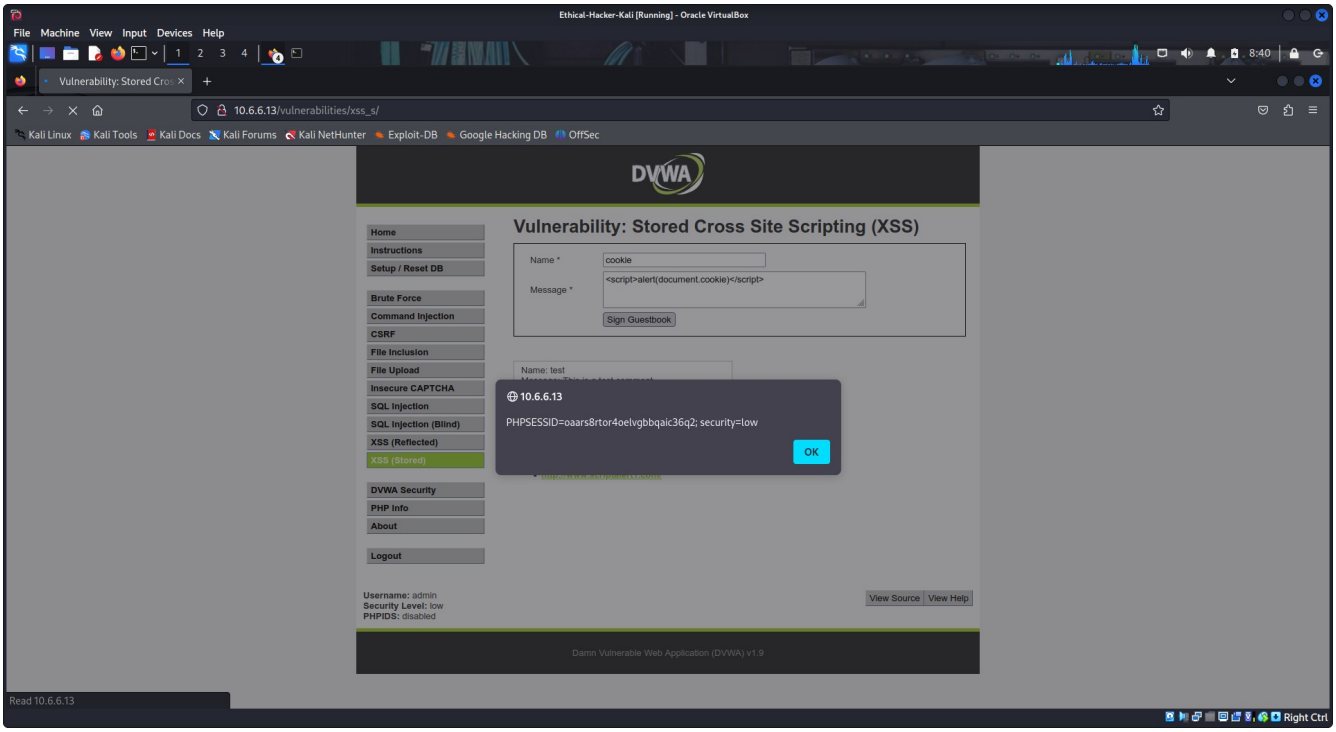
a. Select **DVWA Security** in the left menu and select **Low** in the Security Level options dropdown. Click **Submit**.
b. Select **XSS (Stored)** in the left menu.
c. Type the string **cookie** in the **Name*** field and type the following message in the **Message *** field. click **Sign Guestbook**.

```
<script>alert(document.cookie)</script>
```

A popup box with the cookie will be presented. This is a cookie that PHP uses to keep of track of running sessions.

An exploit could modify the XSS script to have the cookie sent to another destination rather than just displaying it.

d. Click **OK** in the pop-up box.



When conducting an ethical hacking test of a client web application, what does it tell you if the application is vulnerable to XSS at the low, medium, or high level? **It should tell you that the application has serious security vulnerabilities and that there are likely other vulnerabilities present. The web application should be evaluated for more damaging vulnerabilities.**