

Cross Site Scripting (XSS)



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

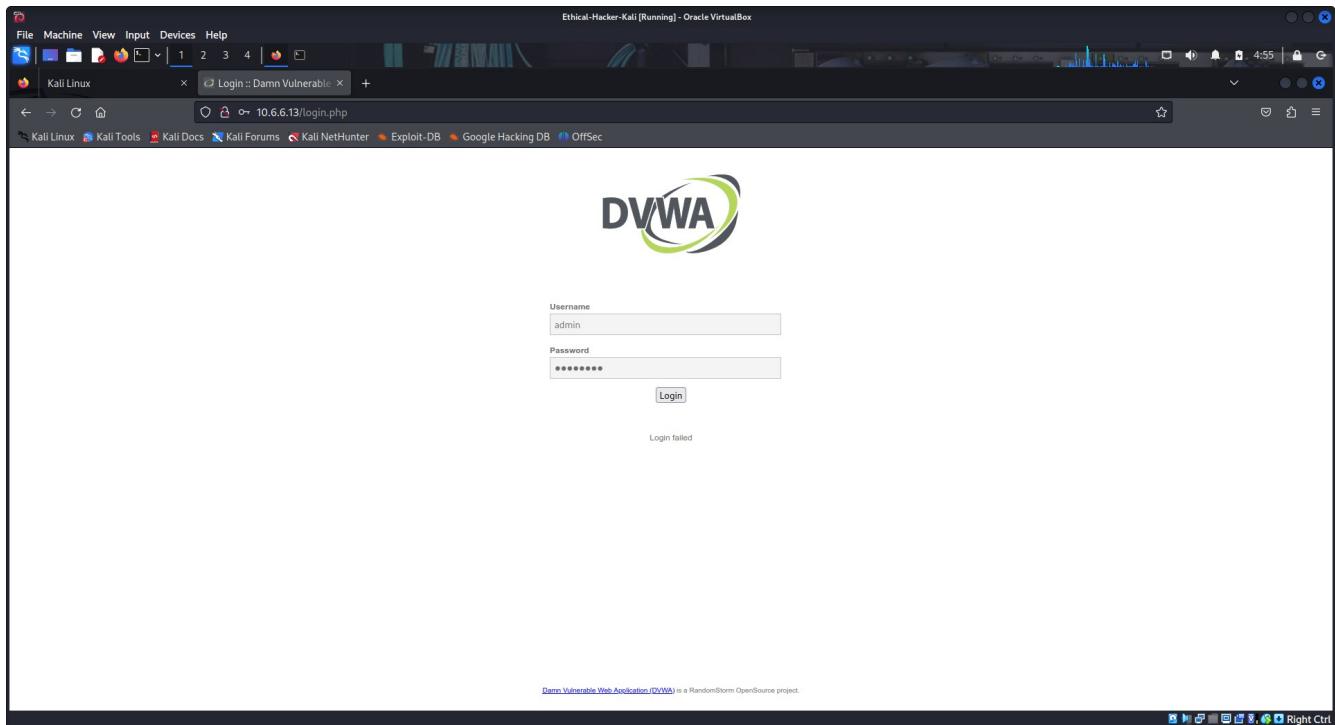
Instructions

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.

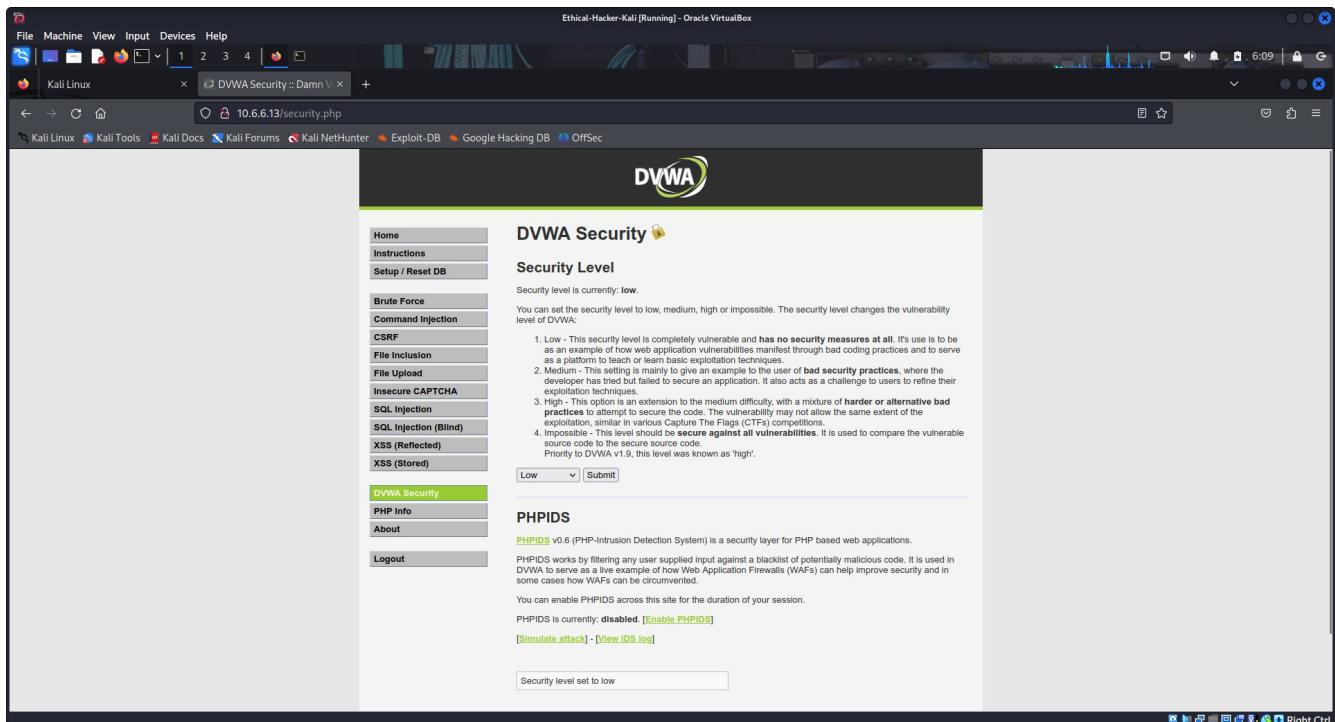
Step 1: Log into DVWA.

- 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.

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



- Select XSS (Reflected) from the left menu.
- Type the string **Reflected_Test** in the What's your name? box and click Submit.

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

The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The address bar shows the URL "http://10.6.6.13/vulnerabilities/xss_r/?name=Reflected_Test#". The main content is the DVWA logo and the title "Vulnerability: Reflected Cross Site Scripting (XSS)". On the left is a sidebar with various menu items. In the center, there is a form with a text input field containing "Hello Reflected_Test". Below the form is a "More Information" section with several links. At the bottom, it says "Username: admin Security Level: low PHPIDS: disabled" and "View Source | View Help". The footer indicates "Damn Vulnerable Web Application (DVWA) v1.9".

- 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.

The screenshot shows a browser window with the developer tools open, specifically the "Elements" tab under "Tools". The URL in the address bar is "http://10.6.6.13/vulnerabilities/xss_r/?name=Reflected_Test#". The page content is displayed in a large pre-tag. Within this content, the string "Hello Reflected_Test" is highlighted in red, indicating it was found via a search operation. The developer tools interface includes a search bar at the top and various filtering options below the code preview.

```

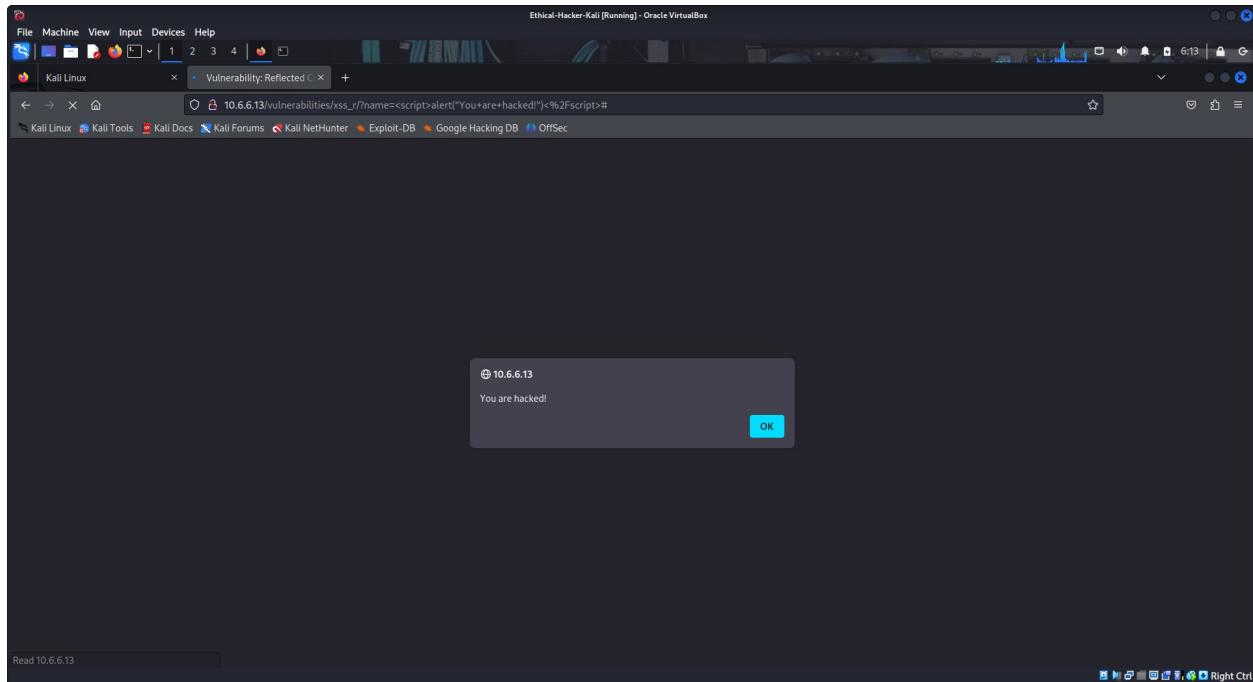
<html>
<head>
<title>Vulnerability: Reflected Cross Site Scripting (XSS)</title>
</head>
<body>
<div><a href="#" onclick="window.location='../../about.php'">About</a></div>
<div><a href="#" onclick="menu8Lock()">Logout</a></div>
<div>What's your name?<br/>
<form name="XSS" action="" method="GET">
<p><input type="text" name="name">
<input type="submit" value="Submit"></p>
</form>
<pre>Hello Reflected_Test</pre>
</div>
<div id="main_body">
<h3>Vulnerability: Reflected Cross Site Scripting (XSS)</h3>
<div class="vulnerable_code_area">
<form name="XSS" action="" method="GET">
<p>What's your name?<br/>
<input type="text" name="name">
<input type="submit" value="Submit"></p>
</form>
<pre>Hello Reflected_Test</pre>
</div>
<div>More Information:</div>
<ul>
<li><a href="https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)" target="_blank">https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)</a></li>
<li><a href="https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet" target="_blank">https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet</a></li>
<li><a href="https://en.wikipedia.org/wiki/Cross-site_scripting" target="_blank">https://en.wikipedia.org/wiki/Cross-site_scripting</a></li>
<li><a href="http://hiderefer.com/http://www.cgisecurity.com/xss-faq.html" target="_blank">http://www.cgisecurity.com/xss-faq.html</a></li>
<li><a href="http://hiderefer.com/http://www.scriptalert1.com/" target="_blank">http://www.scriptalert1.com/</a></li>
</ul>
</div>
<br /><br />
</body>
</html>

```

- 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**.

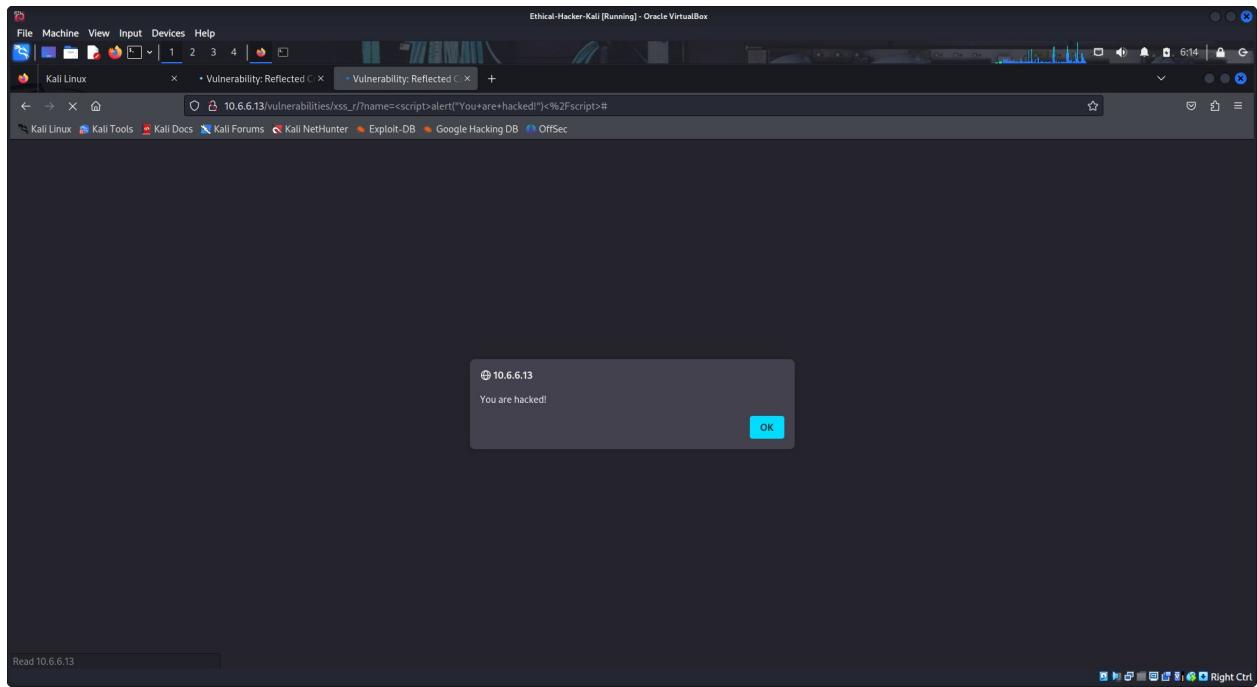
```
<script>alert("You are hacked!")</script>
```

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**.

The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA Security page at 10.6.6.13/security.php. The left sidebar menu has 'XSS (Reflected)' selected. The main content area displays the DVWA Security interface with sections for 'Security Level' (set to medium), 'PHPIDS' (disabled), and a form for entering a payload. The payload entered is '<script>alert("You are hacked!")</script>'. A message box at the bottom indicates the security level is set to medium.

- Select XSS (Reflected) in the left menu.
- 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.

The screenshot shows a Kali Linux desktop environment with a browser window open to the 'Vulnerability: Reflected Cross Site Scripting (XSS)' page at 10.6.6.13/vulnerabilities/xss_r/?name=<script>alert("You+are+hacked!")<%2Fscript>#. The left sidebar menu has 'XSS (Reflected)' selected. The main content area displays the DVWA interface with a message box containing the reflected XSS payload. Below it, a 'More Information' section lists several links related to XSS. The footer of the page shows the DVWA version as v1.9.

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

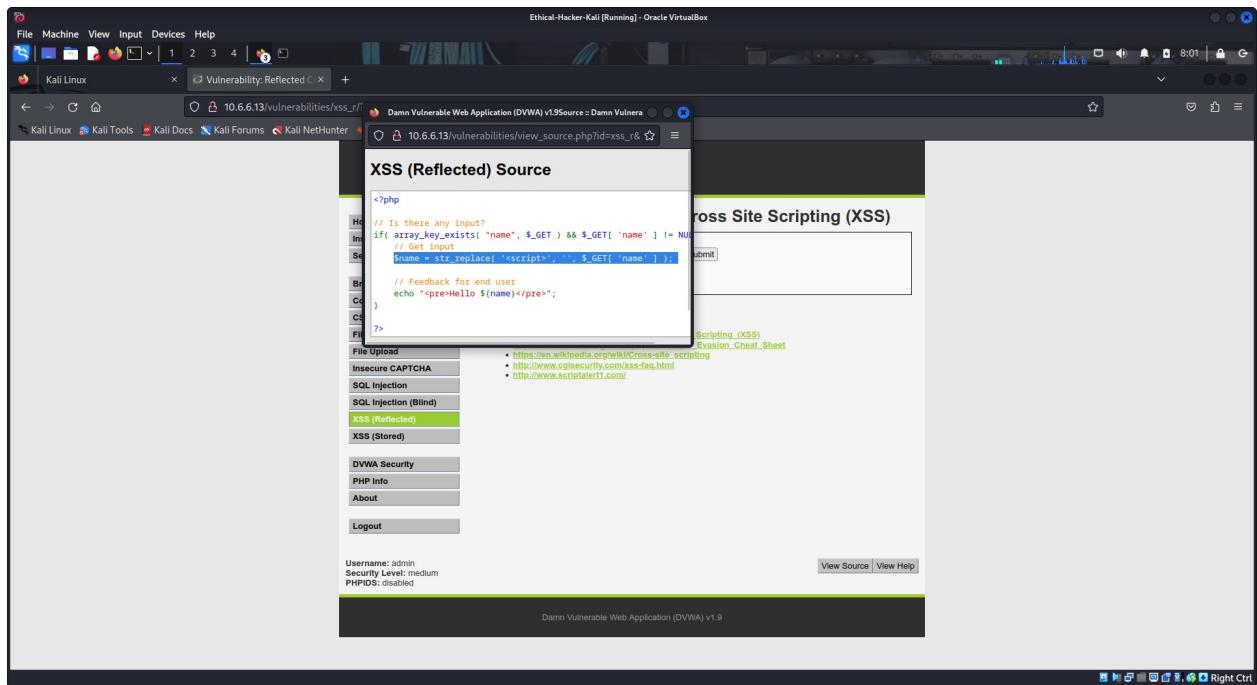
- d. Click the **View Source** button on the bottom right of the page and review the PHP code.

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' ] );
```

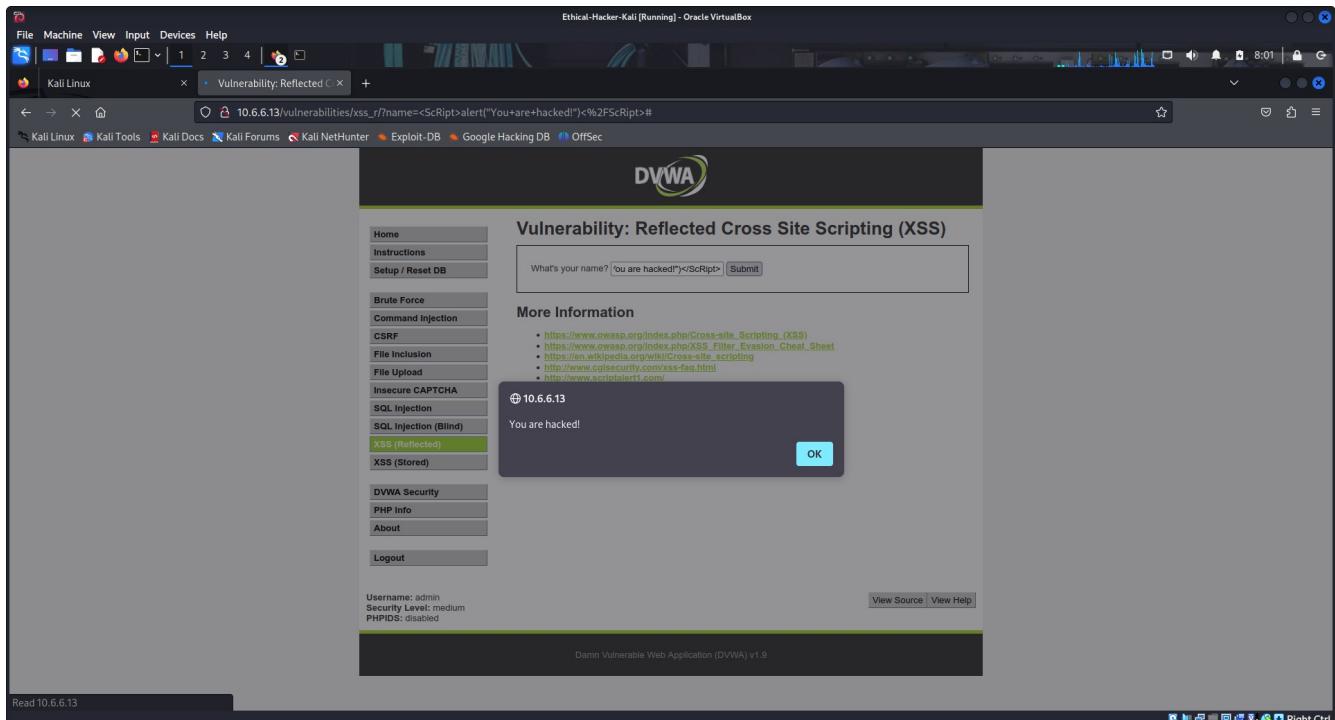
This source code creates a filter, with **str_replace()** function, that removes the **<script>** tag in our payload and replaces it with a null value. This renders the payload script ineffective, so the attack failed, and no popup window is displayed. Because this script is only filtering out **<script>** 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.

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

The screenshot shows a web browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The URL is 10.6.6.13/security.php. The DVWA logo is at the top. The left sidebar menu has "XSS (Reflected)" selected. The main content area is titled "DVWA Security" and "Security Level". It says "Security level is currently: high". Below that is a list of security levels: 1. Low, 2. Medium, 3. High, 4. Impossible. A dropdown menu shows "High" and a "Submit" button. The "PHPIDS" section is also visible.

- 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.

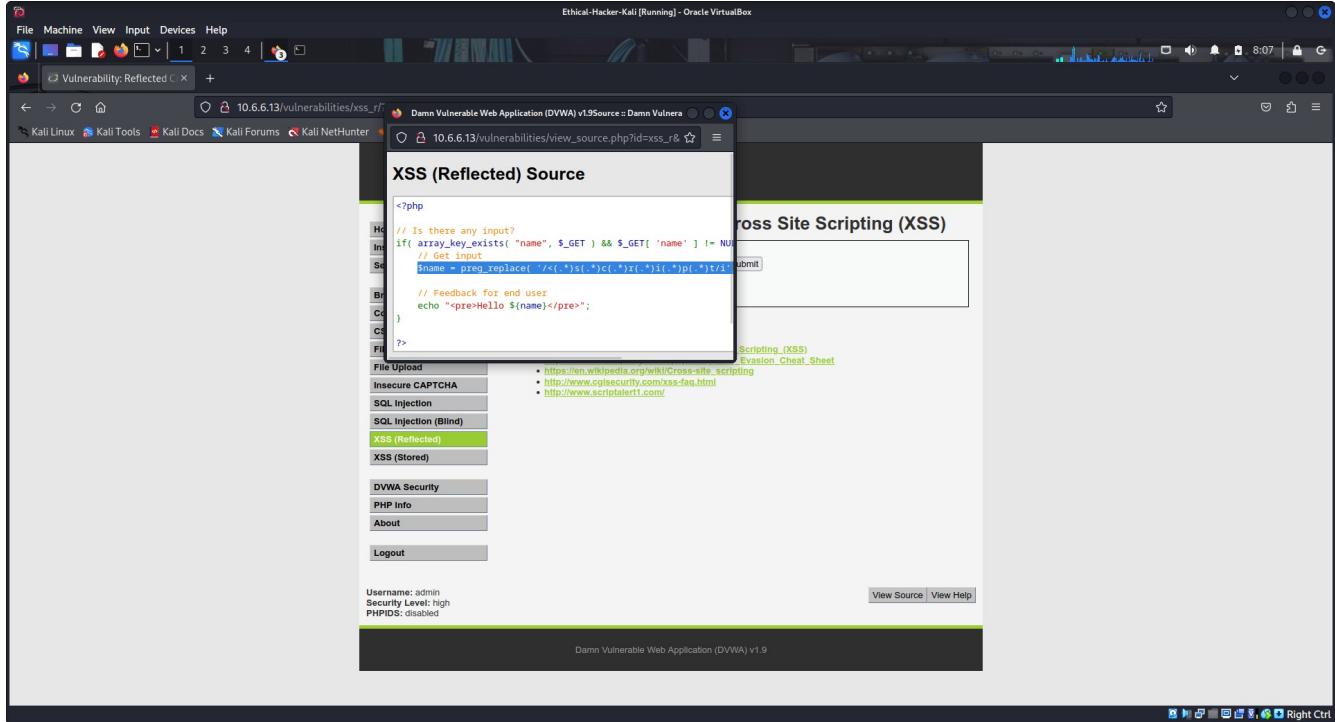
The screenshot shows a web browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The URL is 10.6.6.13/vulnerabilities/xss_r/?name=<ScRipt>alert("You_are_hacked!")<%2FScRipt>#. The DVWA logo is at the top. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It shows a form with "What's your name?" and a "Submit" button. Below it is a message "Hello >". The "More Information" section lists several links about XSS. At the bottom, it shows "Username: admin", "Security Level: high", and "PHPIDS: disabled".

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.**

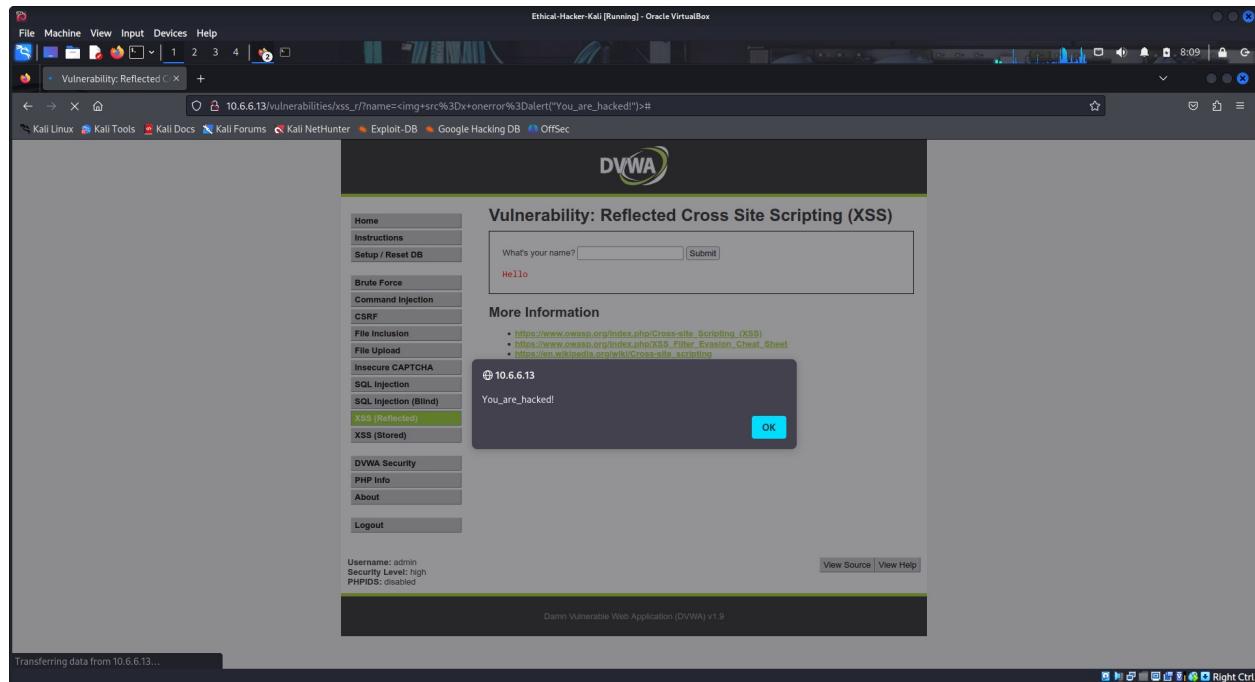
To bypass this filter, we must use another HTML tag instead of **<script>** to attack the site.

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.

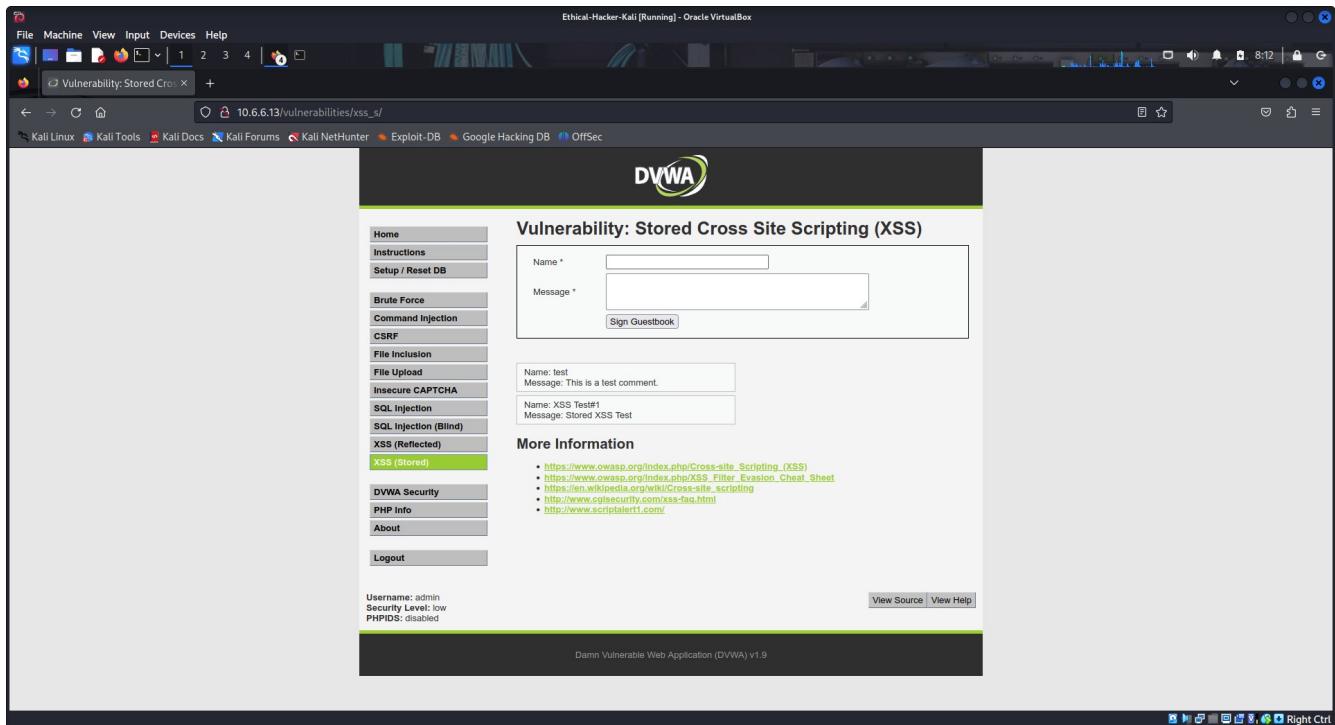
Part 2: Perform Stored Cross Site Scripting Exploits

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.

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

```

File Machine View Input Devices Help
File Machine View Input Devices Help
Vulnerability: Stored Cros... http://10.6.6.13/vulnerabilities/xss_s/
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
View Source View Help
<?php
60 <form method="post" name="guestform" onsubmit="return validate_form(this)">
61   <table width="558" border="0" cellpadding="2" cellspacing="1">
62     <tr>
63       <td width="100">Name </td>
64       <td><input name="txtName" type="text" size="30" maxlength="10"></td>
65     </tr>
66     <tr>
67       <td width="100">Message </td>
68       <td><textarea name="mtgMessage" cols="50" rows="3" maxlength="50"></textarea></td>
69     </tr>
70     <tr>
71       <td width="100">&ampnbsp</td>
72       <td><input name="btnSign" type="submit" value="Sign Guestbook" onClick="return checkForm();"></td>
73     </tr>
74   </table>
75
76 </form>
77
78 </div>
79 <br />
80
81 <div id="guestbook_comments">Name: test<br />Message: This is a test comment.<br /></div>
82 <div id="guestbook_comments">Name: XSS Test#1<br />Message: Stored XSS Test<br /></div>
83 <div id="guestbook_comments">Name: XSS Test#1<br />Message: Stored XSS Test<br /></div>
84
85 <br />
86
87 <h2>More Information</h2>
88 <ul>
89   <li><a href="http://hiderefer.com/http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)" target="_blank">https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)</a></li>
90   <li><a href="http://hiderefer.com/https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet" target="_blank">https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet</a></li>
91   <li><a href="http://hiderefer.com/http://en.wikipedia.org/wiki/Cross-site_scripting" target="_blank">https://en.wikipedia.org/wiki/Cross-site_scripting</a></li>
92   <li><a href="http://hiderefer.com/http://www.cgisecurity.com/xss-faq.html" target="_blank">http://www.cgisecurity.com/xss-faq.html</a></li>
93   <li><a href="http://hiderefer.com/http://www.scriptalert1.com/" target="_blank">http://www.scriptalert1.com/</a></li>
94 </ul>
95
96 <br /><br />
97
98
99
100 </div>
101 <div class="clear">
102 </div>
103
104

```

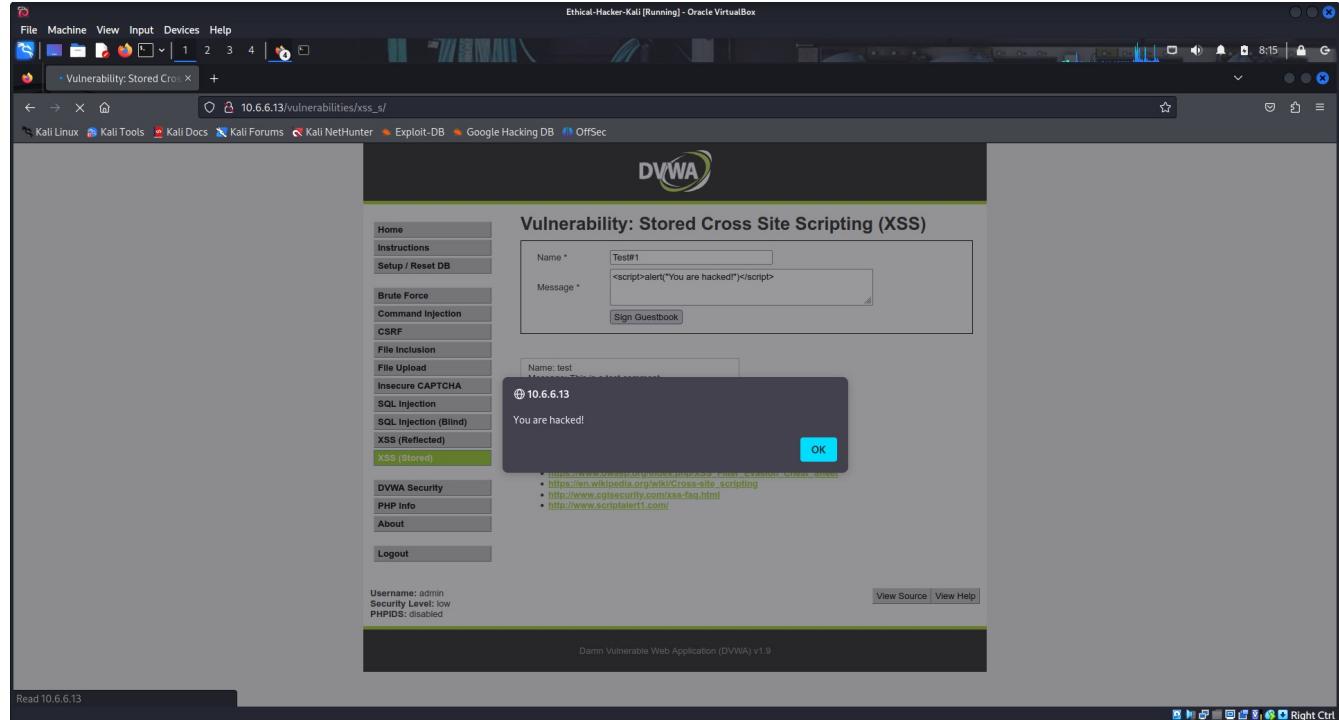
Test#1| ^ Highlight All Match Case Match Diacritics Whole Words 1 of 2 matches

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

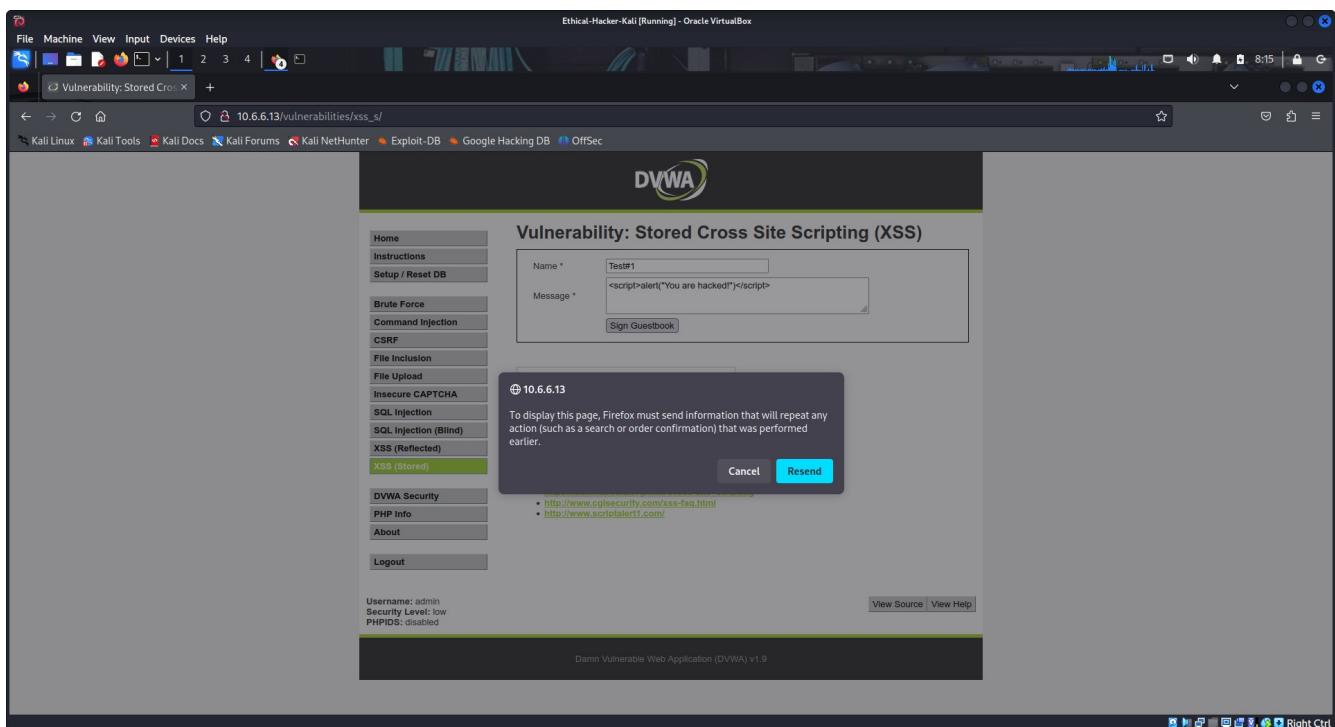
- f. Enter **Test#1** in the **Name *** box and enter the following payload in the **Message *** field and click **Sign Guestbook**.

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

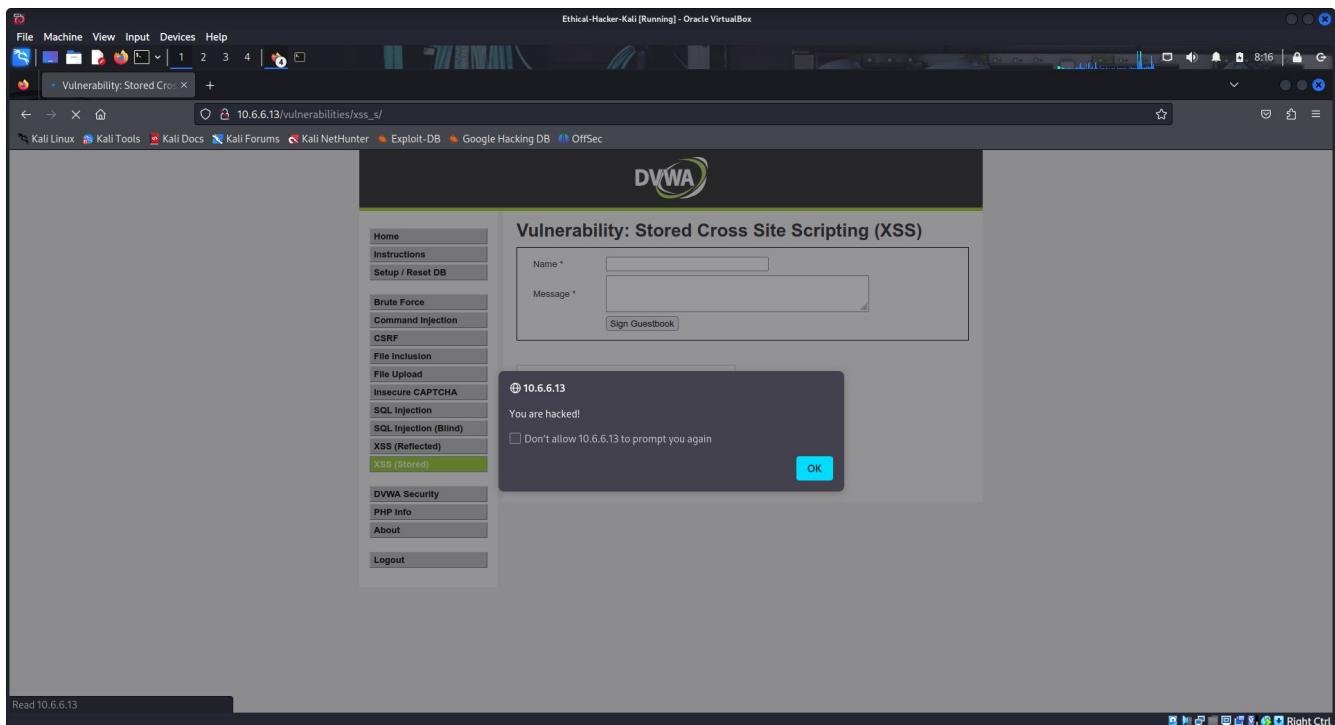
An XSS alert popup box will appear with the words **You are hacked!**. This means the site was vulnerable to stored XSS attacks and we have successfully exploited the vulnerability.



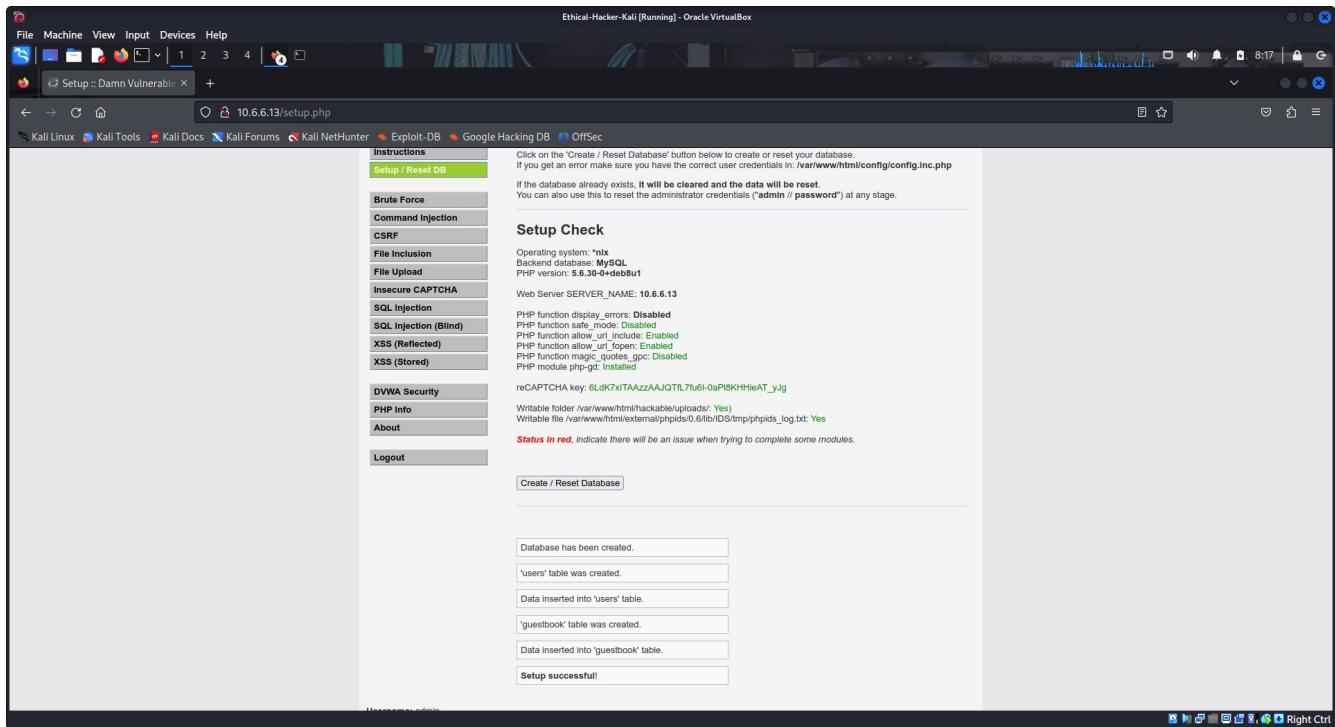
- g. Refresh the page. If alerted, click **Resend** to display the page. The XSS alert popup box will appear again.



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**.

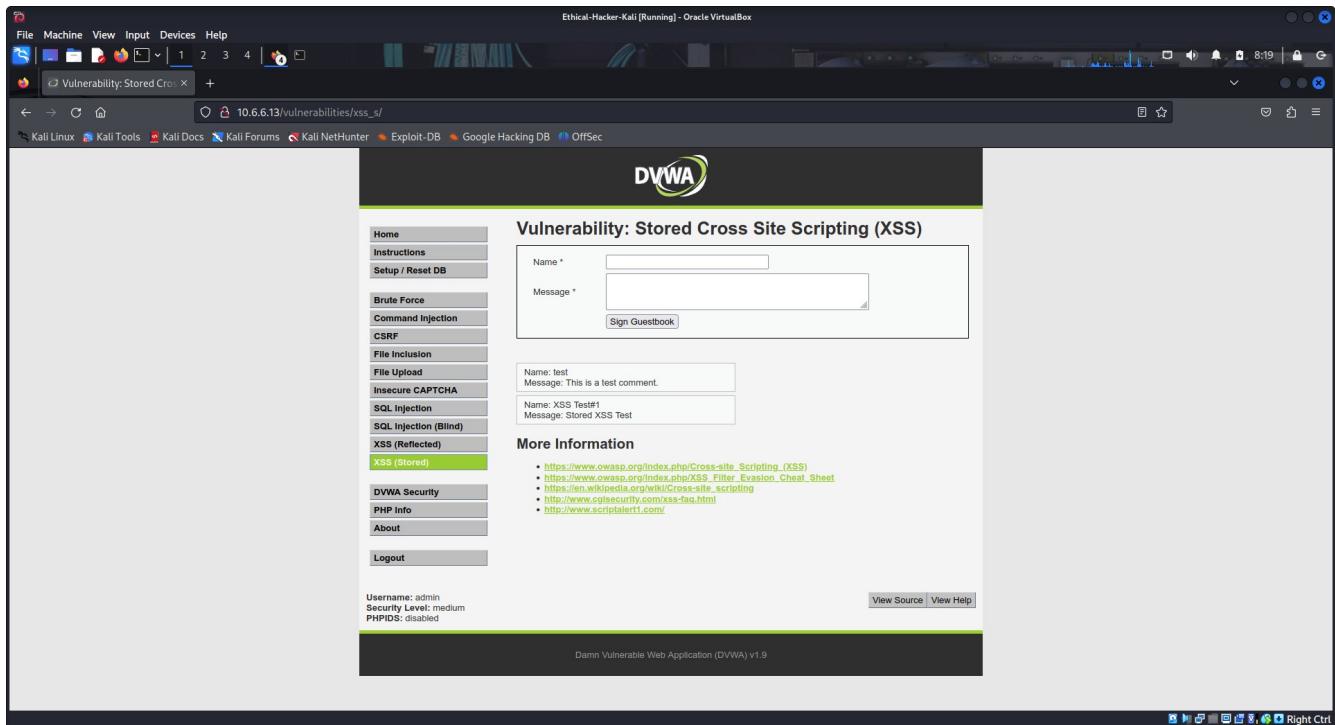


Step 2: Perform a Stored XSS attack at Medium security level.

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.

The screenshot shows the browser developer tools with the 'view-source' tab selected, displaying the HTML source code of the DVWA Stored XSS page. The source code includes the form structure with 'Name' and 'Message' fields, and the examples of stored XSS attacks. A search bar at the bottom of the developer tools window has 'Test#1' entered, and the results show two matches found in the source code.

```

<form method="post" name="guestform" onsubmit="return validate_form(this)">
    <table width="558" border="0" cellpadding="2" cellspacing="1">
        <tr>
            <td width="100">Name </td>
            <td><input name="txtName" type="text" size="30" maxlength="10"></td>
        </tr>
        <tr>
            <td width="100">Message </td>
            <td><textarea name="txtMessage" cols="50" rows="3" maxlength="50"></textarea></td>
        </tr>
        <tr>
            <td width="100">&ampnbsp</td>
            <td><input name="btnSign" type="submit" value="Sign Guestbook" onClick="return checkForm();"></td>
        </tr>
    </table>
</form>
<br />
<div id="guestbook_comments">Name: test<br />Message: This is a test comment.<br /></div>
<div id="guestbook_comments">Name: XSS Test#1<br />Message: Stored XSS Test<br /></div>
<div id="guestbook_comments">Name: XSS Test#1<br />Message: Stored XSS Test<br /></div>
<br />
<h2>More Information:</h2>
<ul>
    <li><a href="http://hiderefer.com/http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)" target="_blank">https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)</a></li>
    <li><a href="http://hiderefer.com/https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet" target="_blank">https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet</a></li>
    <li><a href="http://hiderefer.com/http://en.wikipedia.org/wiki/Cross-site_scripting" target="_blank">https://en.wikipedia.org/wiki/Cross-site_scripting</a></li>
    <li><a href="http://hiderefer.com/http://www.cgisecurity.com/xss-faq.html" target="_blank">http://www.cgisecurity.com/xss-faq.html</a></li>
    <li><a href="http://hiderefer.com/http://www.scriptalert1.com/" target="_blank">http://www.scriptalert1.com/</a></li>
</ul>
<br /><br />
<div class="clear">
</div>

```

- 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.

The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The address bar shows the URL "10.6.6.13/vulnerabilities/xss_s/". The main content is the DVWA application's "Vulnerability: Stored Cross Site Scripting (XSS)" page. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), and XSS (Stored). The "XSS (Stored)" option is highlighted. The main form has two input fields: "Name *" and "Message *". Below the form, a list of previous entries shows that the input was sanitized. The last entry in the list is "Name: Test#1" and "Message: alert('You are hacked!')". At the bottom right of the page are "View Source" and "View Help" buttons. The footer of the page reads "Damn Vulnerable Web Application (DVWA) v1.9".

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.

The screenshot shows a Kali Linux desktop environment with a browser window open to DVWA. The URL is 10.6.6.13/vulnerabilities/view_source.php?id=xss_s&security=medium. A modal window titled "XSS (Stored) Source" displays the PHP source code. The code includes logic for handling POST variables and performing input sanitization using addslashes() and str_replace() functions. The XSS payload is injected into the \$name variable. The browser output shows the injected script being executed, resulting in an alert box.

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.

The screenshot shows a Kali Linux desktop environment with a browser window titled 'Vulnerability: Stored Cross Site Scripting (XSS)' at the URL http://10.6.6.13/vulnerabilities/xss_s/. The DVWA logo is at the top. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), and XSS (Stored). The 'XSS (Stored)' option is selected. The main content area displays a guestbook form with fields for 'Name *' and 'Message *'. Below the form, four entries are shown in a table:

Name: test	Message: This is a test comment.
Name: XSS Test#1	Message: Stored XSS Test
Name: XSS Test#1	Message: Stored XSS Test
Name: Test#1	Message: alert('You are hacked!')

Below the table, a link points to [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)). The bottom of the screen shows the Web Developer Tools window with the CSS tab selected, showing styles for the 'main.css' file.

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.

The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The URL is 10.6.6.13/vulnerabilities/xss_s/. The DVWA logo is at the top. On the left, a sidebar menu includes Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), and XSS (Stored). The XSS (Stored) option is highlighted. The main content area displays the "Vulnerability: Stored Cross Site Scripting (XSS)" page. A form has "Name" set to "<script>alert('You are hacked!')</script>" and "Message" set to "hacker". A "Sign Guestbook" button is present. Below the form, a message box shows "10.6.6.13" and "You are hacked!". An "OK" button is visible. A "More Information" section lists several XSS links. At the bottom, it says "Username: admin", "Security Level: medium", and "PHPIDS: disabled". There are "View Source" and "View Help" links. The footer reads "Damn Vulnerable Web Application (DVWA) v1.9".

1. 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**.

The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The URL is 10.6.6.13/setup.php. The DVWA sidebar menu is visible. The main content area shows the "Database Setup" page. It includes a "Setup Check" section with system details: Operating system: "mx", Backend database: "MySQL", PHP version: "5.6.30-0+deb8u1". It also shows "Web Server SERVER_NAME: 10.6.6.13" and "reCAPTCHA key: 6LQK7xitAAza2AJQTL7u6I-oPjBKHieAT_yJg". A note states: "Click on the 'Create / Reset Database' button below to create or reset your database. If you get an error make sure you have the correct user credentials in: /var/www/html/config/config.inc.php If the database already exists, it will be cleared and the data will be reset. You can also use this to reset the administrator credentials ('admin // password') at any stage." Below this is a "Setup Check" table with various PHP settings. A red status message "Status in red. Indicate there will be an issue when trying to complete some modules." is shown. At the bottom, a "Create / Reset Database" button is present. A success message box shows "Database has been created.", "'users' table was created.", "Data inserted into 'users' table.", "'guestbook' table was created.", "Data inserted into 'guestbook' table.", and "Setup successful!".

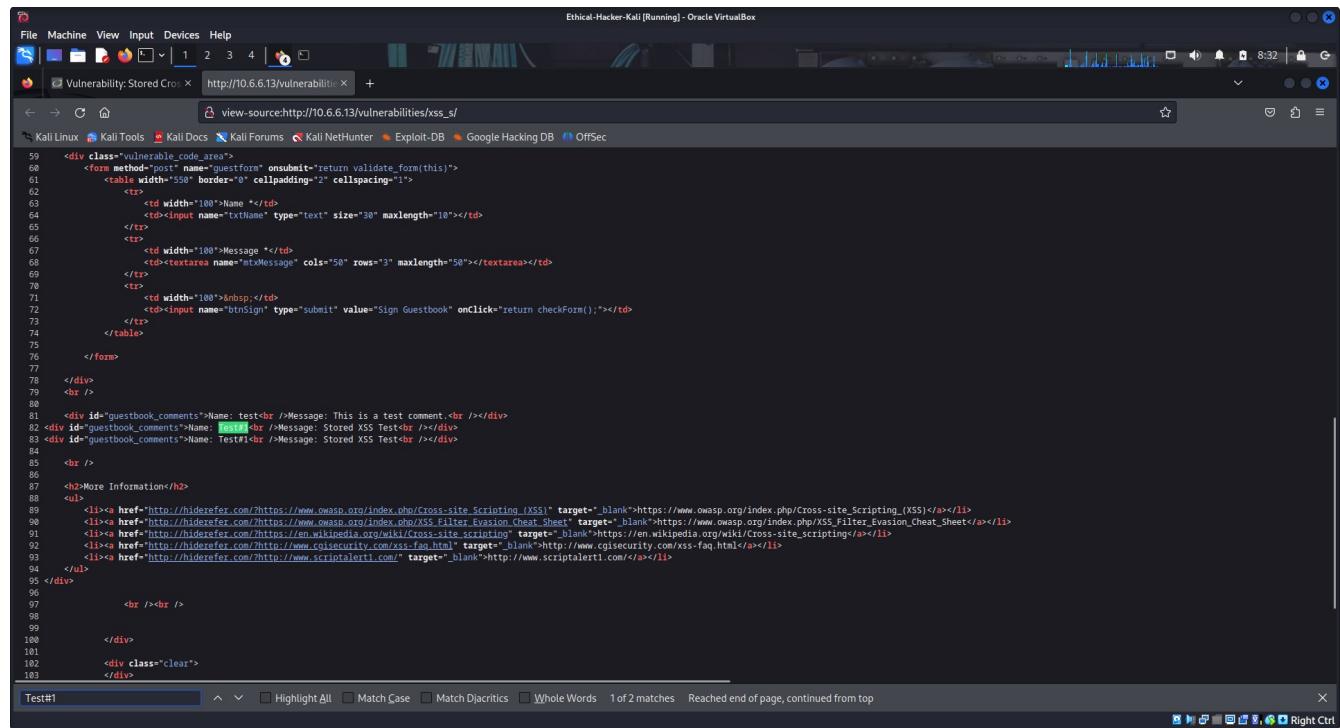
Step 3: Perform a Stored XSS attack at High security level.

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.



The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The address bar shows "http://10.6.6.13/vulnerabilities/xss_s/". The page content displays the source code of a guestbook form. The source code includes several comments and user input fields. The payload "Test#1" is visible in the source code, indicating it was stored in the database. The browser's status bar at the bottom shows "Test#1" in the search field, along with other search options like "Highlight All", "Match Case", etc.

```
File Machine View Input Devices Help
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15 F16 F17 F18 F19 F20 F21 F22 F23 F24 F25 F26 F27 F28 F29 F30 F31 F32 F33 F34 F35 F36 F37 F38 F39 F40 F41 F42 F43 F44 F45 F46 F47 F48 F49 F50 F51 F52 F53 F54 F55 F56 F57 F58 F59 F60 F61 F62 F63 F64 F65 F66 F67 F68 F69 F70 F71 F72 F73 F74 F75 F76 F77 F78 F79 F80 F81 F82 F83 F84 F85 F86 F87 F88 F89 F90 F91 F92 F93 F94 F95 F96 F97 F98 F99 F100 F101 F102 F103
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
View-source:http://10.6.6.13/vulnerabilities/xss_s/
view-source:http://10.6.6.13/vulnerabilities/xss_s/
59 <div class="vulnerable_code_area">
60   <form method="post" name="guestform" onsubmit="return validate_form(this)">
61     <table width="550" border="0" cellpadding="2" cellspacing="1">
62       <tr>
63         <td width="100">Name *</td>
64         <td><input name="txtName" type="text" size="30" maxlength="10"></td>
65       </tr>
66       <tr>
67         <td width="100">Message *</td>
68         <td><textarea name="txtMessage" cols="50" rows="3" maxlength="50"></textarea></td>
69       </tr>
70       <tr>
71         <td width="100">&ampnbsp</td>
72         <td><input name="btnSign" type="submit" value="Sign Guestbook" onClick="return checkForm();"></td>
73       </tr>
74     </table>
75   </form>
76 </div>
77 <br />
78 <br />
79 <br />
80 <div id="guestbook_comments">Name: test<br />Message: This is a test comment.<br /></div>
81 <div id="guestbook_comments">Name: Test#1<br />Message: Stored XSS Test<br /></div>
82 <div id="guestbook_comments">Name: Test#1<br />Message: Stored XSS Test<br /></div>
83 <br />
84 <br />
85 <h2>More Information:</h2>
86 <ul>
87   <li><a href="http://hiderefer.com/?https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)" target="_blank">https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)</a></li>
88   <li><a href="http://hiderefer.com/?https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet" target="_blank">https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet</a></li>
89   <li><a href="http://hiderefer.com/?https://en.wikipedia.org/w/index.php?title=Cross-site_scripting" target="_blank">https://en.wikipedia.org/w/index.php?title=Cross-site_scripting</a></li>
90   <li><a href="http://hiderefer.com/?https://en.wikipedia.org/w/index.php?title=Cross-site_scripting" target="_blank">https://en.wikipedia.org/w/index.php?title=Cross-site_scripting</a></li>
91   <li><a href="http://hiderefer.com/?https://cgisecurity.com/xss-faq.html" target="_blank">http://www.cgisecurity.com/xss-faq.html</a></li>
92   <li><a href="http://hiderefer.com/?https://www.scriptalert1.com/" target="_blank">http://www.scriptalert1.com/</a></li>
93   <li><a href="http://hiderefer.com/?https://www.scriptalert1.com/" target="_blank">http://www.scriptalert1.com/</a></li>
94 </ul>
95 </div>
96 <br /><br />
97 <br />
98 <br />
99 <br />
100 <div class="clear">
101 </div>
102 <div class="clear">
103 </div>
```

- 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.

The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA 'Vulnerability: Stored Cross Site Scripting (XSS)' page. The URL is 10.6.6.13/vulnerabilities/xss_s/. The page displays a guestbook form with several entries. On the right, a modal window titled 'XSS (Stored) Source' shows the PHP source code for the application's logic:

```

<?php
if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = mysql_real_escape_string( $message );
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = preg_replace( '/<(.*)>/i', '', $name );
    $name = mysql_real_escape_string( $name );

    // Update database
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' )";
    $result = mysql_query( $query ) or die( '<pre>' . mysql_error() . '</pre>' );
}

//mysql_close();
?

```

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.

The screenshot shows a Kali Linux desktop environment with a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The browser is displaying the DVWA (Damn Vulnerable Web Application) "Vulnerability: Stored Cross Site Scripting (XSS)" page at the URL http://10.6.6.13/vulnerabilities/xss_s/. The page contains a guestbook form with fields for Name and Message, and a "Sign Guestbook" button. Below the form, there is a list of previous entries:

- Name: test
Message: This is a test comment.
- Name: Test#1
Message: Stored XSS Test
- Name: Test#1
Message: Stored XSS Test
- Name: Test#1
Message: alert('You are hacked!')

The developer tools (F12) are open, showing the DOM structure of the guestbook form. The payload `<svg onload=alert("You_are_hacked!")>` has been entered into the Name field. The browser's status bar indicates the URL [https://www.w3schools.com/html/index.php/Cross-site_Scripting_\(XSS\)](https://www.w3schools.com/html/index.php/Cross-site_Scripting_(XSS)).

- 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!")>
```

- 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.

The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The address bar shows the URL 10.6.6.13/vulnerabilities/xss_s/. The main content is the DVWA "Vulnerability: Stored Cross Site Scripting (XSS)" page. On the left, a sidebar menu lists various vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), and XSS (Stored). The "XSS (Stored)" option is selected. Below the menu, there's a form with fields for "Name" and "Message", and a "Sign Guestbook" button. A modal dialog box appears with the message "10.6.6.13 You_are_hacked!" and an "OK" button. Another smaller modal shows the message "Message: alert('You are hacked!')". At the bottom, there's a "More Information" section with links to OWASP XSS resources and a "Logout" link.

- 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**.

The screenshot shows a browser window titled "Ethical-Hacker-Kali [Running] - Oracle VirtualBox". The address bar shows the URL 10.6.6.13/setup.php. The main content is the DVWA "Database Setup" page. The left sidebar menu is identical to the previous screenshot, with "XSS (Stored)" selected. The main area contains a "Setup Check" section with system information and a "Create / Reset Database" button. Below this, a series of success messages are listed in a box: "Database has been created.", "'users' table was created.", "Data inserted into 'users' table.", "'guestbook' table was created.", "Data inserted into 'guestbook' table.", and "Setup successful!".

Step 4: Perform a stored iframe exploit.

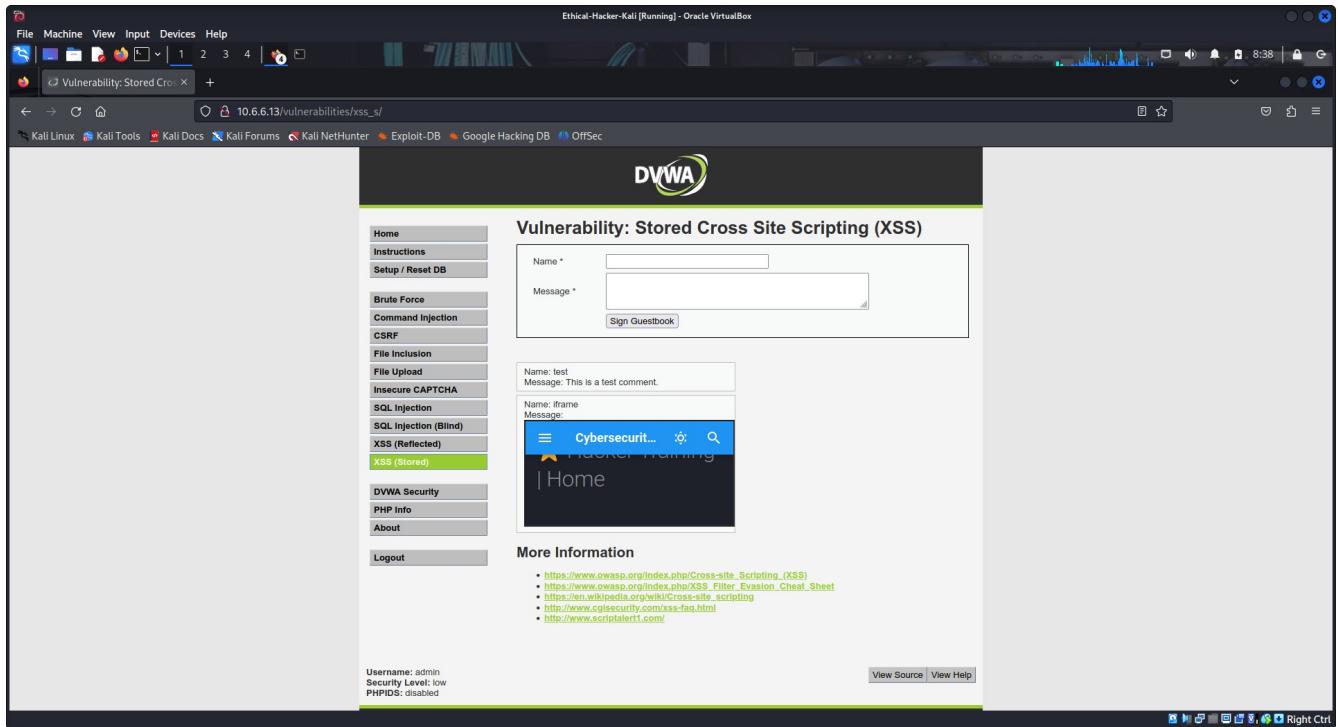
- Select **DVWA Security** in the left menu and select **Low** in the Security Level dropdown. Click **Submit**.
- Select **XSS (Stored)** in the left menu.

- 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>
```

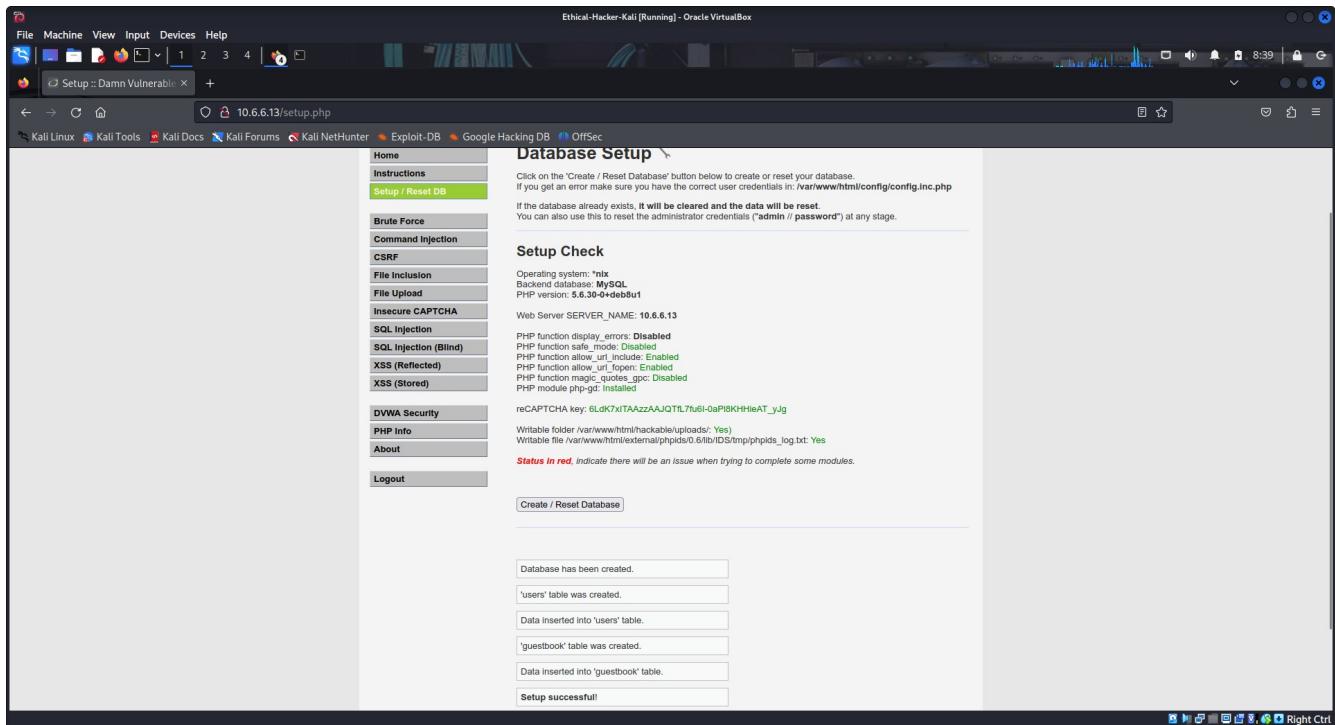
The H4cker website should now be displayed under the iframe test message.

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



A screenshot of a web browser window showing the DVWA (Damn Vulnerable Web Application) interface. The URL in the address bar is 10.6.6.13/vulnerabilities/xss_s/. The main content area displays a form titled "Vulnerability: Stored Cross Site Scripting (XSS)". The "Name" field contains "iframe" and the "Message" field contains "<iframe src='http://h4cker.org'></iframe>". Below the form, there is a preview of the stored XSS payload: "Name: test" and "Message: This is a test comment." followed by "Name: iframe" and "Message: Cybersecurity". A screenshot of the H4cker website is shown within the message box, displaying the text "Cybersecurity" and "Hacker Training". At the bottom of the page, there is a "More Information" section with links to various XSS resources and a "Logout" link. The status bar at the bottom shows "Username: admin", "Security Level: low", and "PHPIDS: disabled".

- 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.

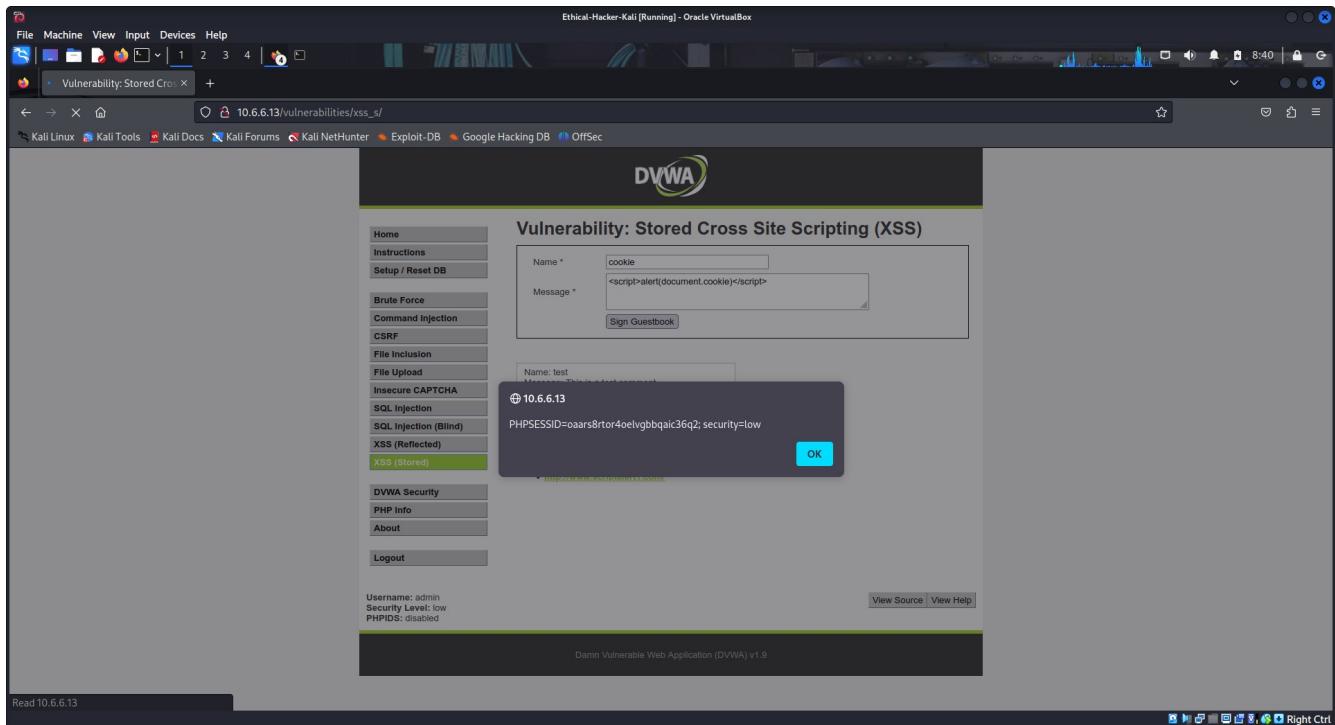
- Select **DVWA Security** in the left menu and select **Low** in the Security Level options dropdown. Click **Submit**.
- Select **XSS (Stored)** in the left menu.
- Type the string **cookie** in the **Name*** field and type the following message in the **Message *** field. click **Sign Guestbook**.

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

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

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

- 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.**