XSS Stored Attacks on DVWA

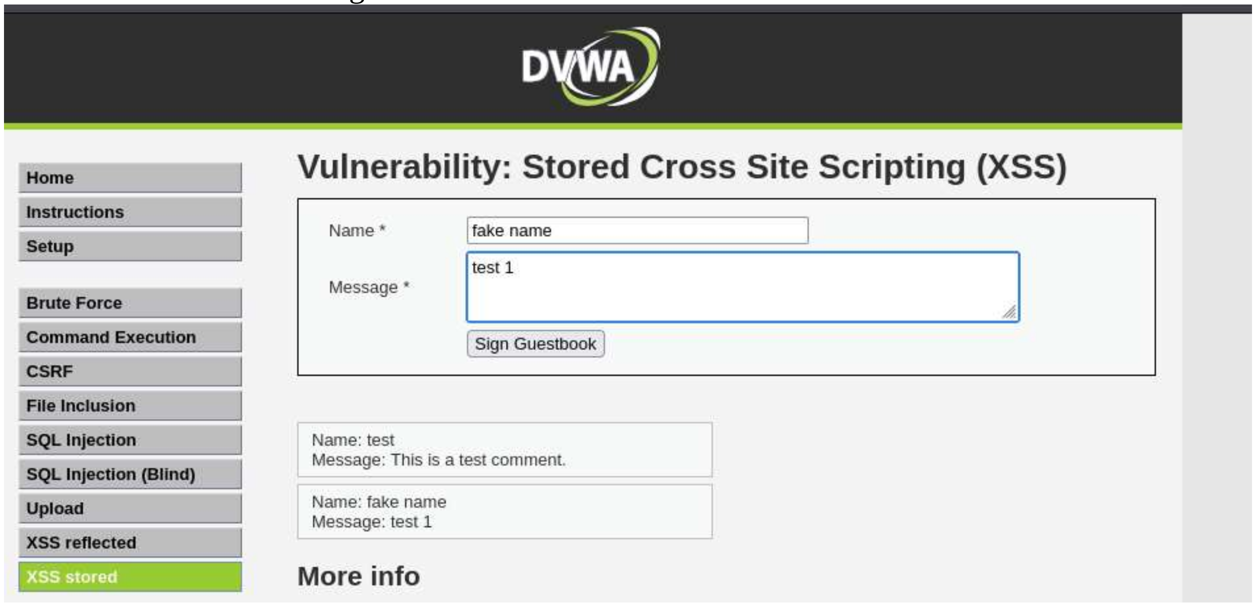
Prepared by Md Jobarul Islam

Md Jobarul Islam
Mirpur, Dhaka
email: jobarulislam1203@gmail.com
linkedIn: https://www.linkedin.com/in/jobarulislam/
GitHub: https://github.com/jobarulislam
Phone: +880 1312-678017

Login to the dvwa and select XSS reflected :

Low security XSS Stored Attack

1st Observe how its working :



Vulnerability: Stored Cross Site Scripting (XSS)

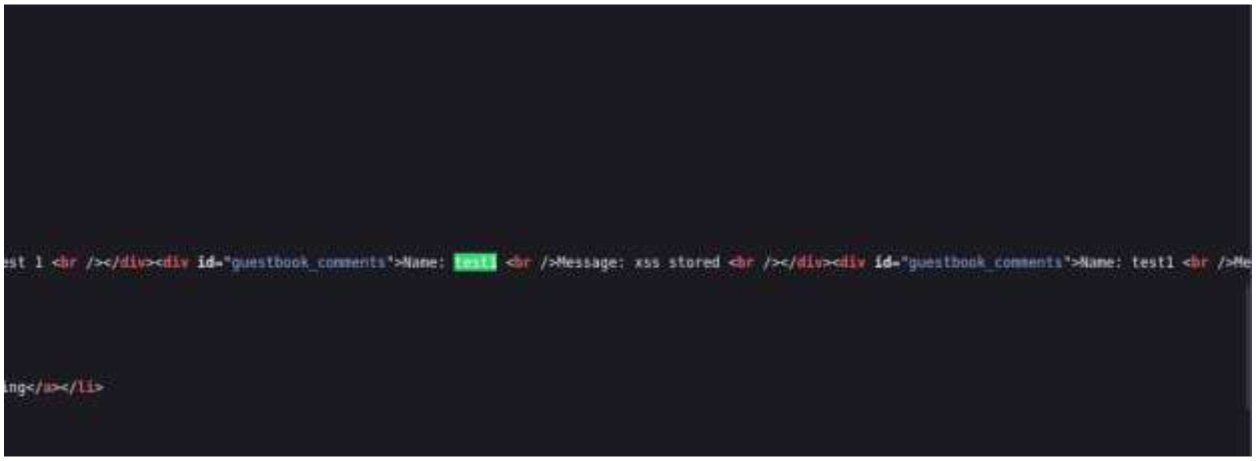
Name * fake name

Message * test 1

Name: test
Message: This is a test comment.

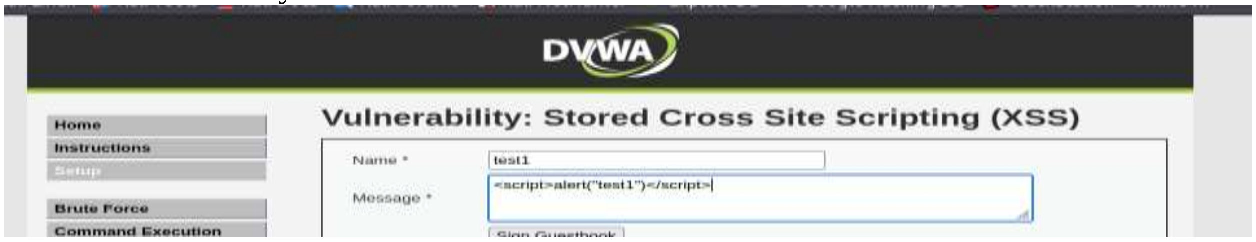
Name: fake name
Message: test 1

Use (Ctrl + u) for open source code:
(Ctrl + f) for search [hello reflected_text]
Back to Render window by (Ctrl + w)



```
est 1 <br /></div><div id="guestbook_comments">Name: test1 <br />Message: xss stored <br /></div><div id="guestbook_comments">Name: test1 <br />Me

ing</a></li>
```

Submit [<script>alert("this is a test! ")</script>] code in input box for check for which one work Xss valnareblity:

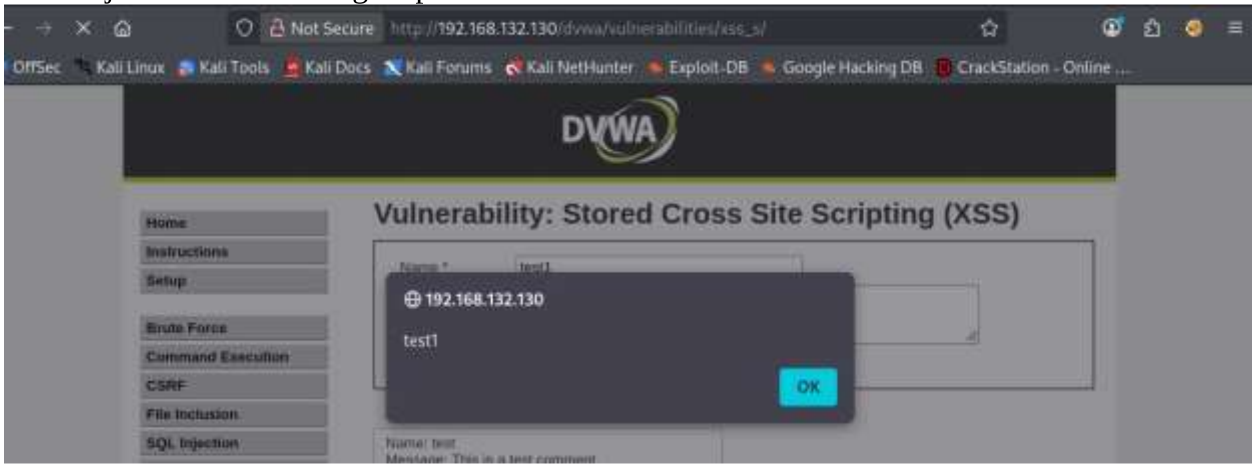


Vulnerability: Stored Cross Site Scripting (XSS)

Name * test1

Message * <script>alert("test1")</script>

Code Inject work on massage input box so this is vulnerable :


Not Secure http://192.168.132.130/dvwa/vulnerabilities/xss_s/
OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB CrackStation - Online ...
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection

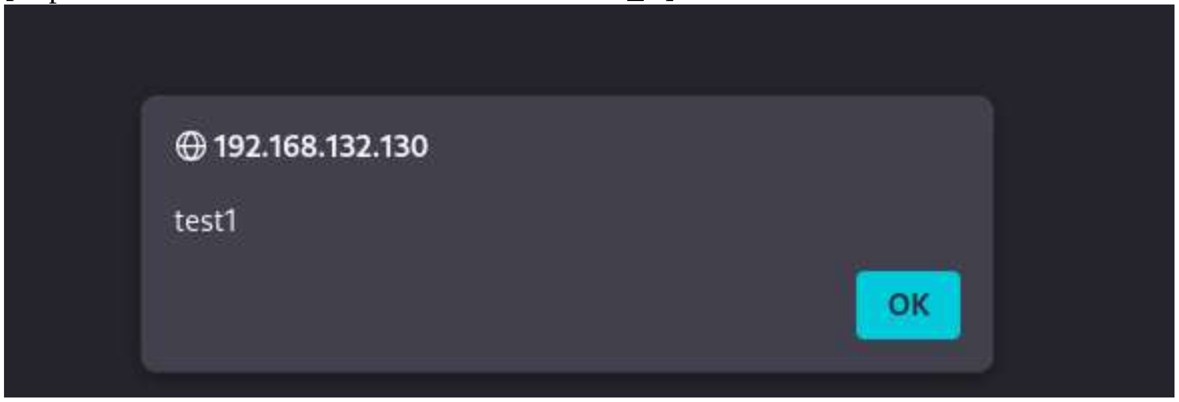
Vulnerability: Stored Cross Site Scripting (XSS)

192.168.132.130

test1

OK

Url of this page :
[http://192.168.132.130/dvwa/vulnerabilities/xss_s/]

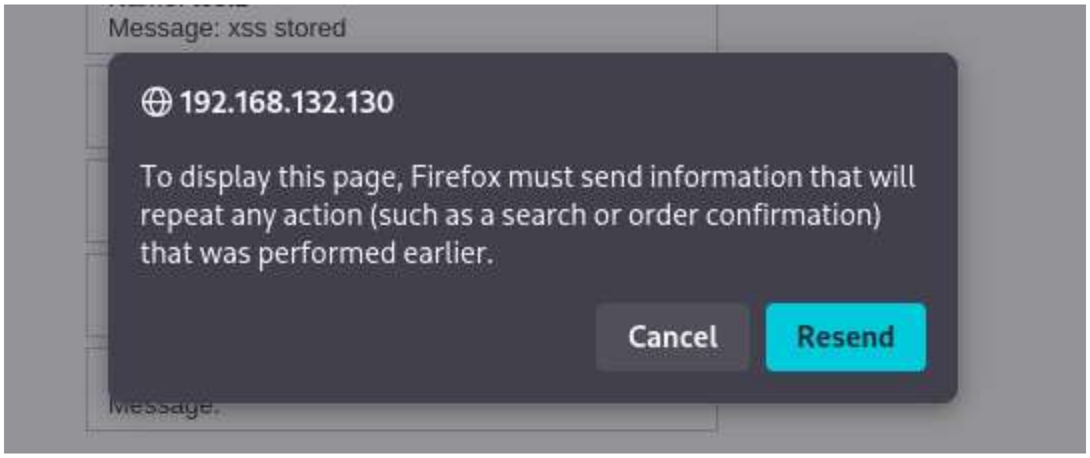


192.168.132.130

test1

OK

Its seems that this page have vulnerability and I found it. So it’s can by compromised by attacker of XSS Stored attack.
Every time resend the alert will appear again and again because malicious code/ payload stored in the server.



Message: xss stored

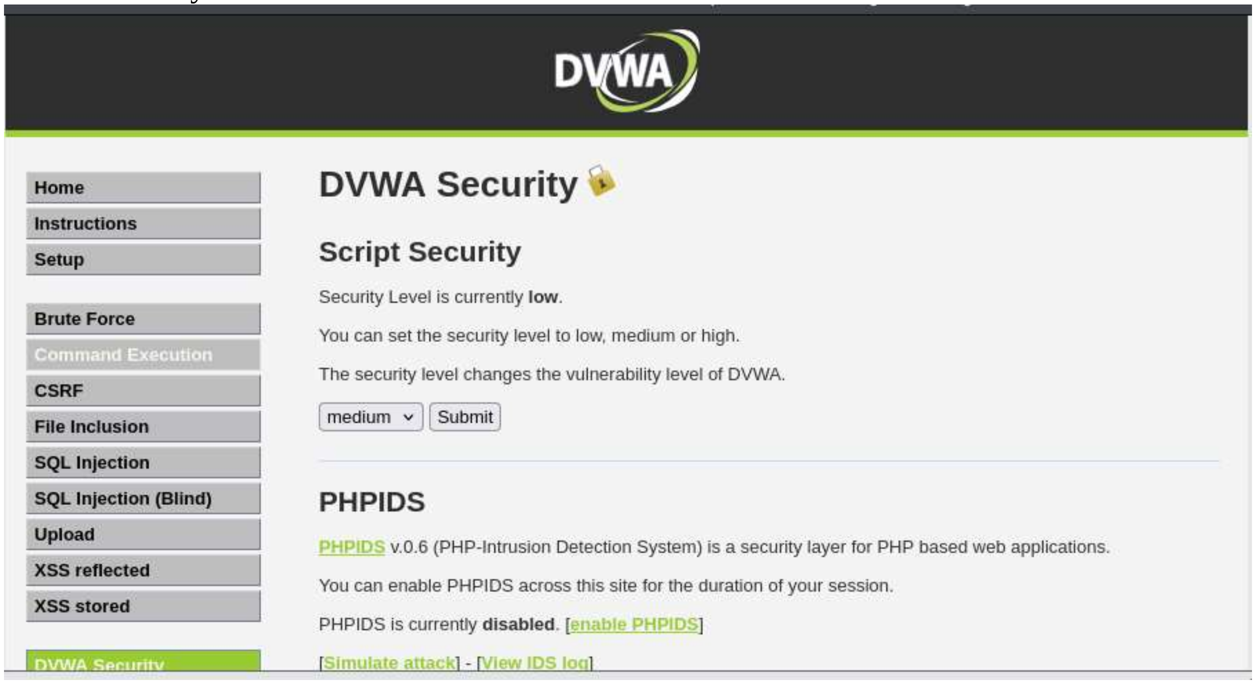
192.168.132.130

To display this page, Firefox must send information that will repeat any action (such as a search or order confirmation) that was performed earlier.

Cancel Resend

Medium security XSS Stored Attack

Submit security level low to medium:



DVWA

Home
Instructions
Setup
Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security

DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

medium [Submit]

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

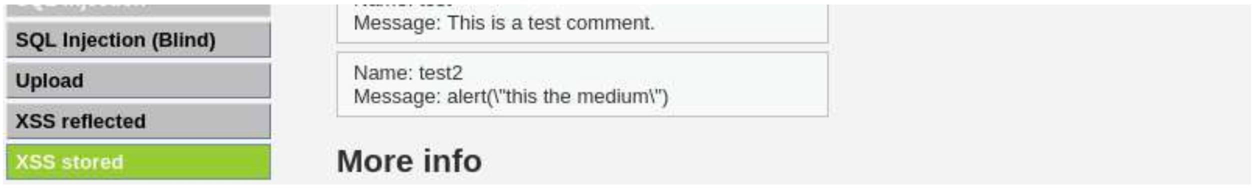
You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled**. [enable PHPIDS]

[Simulate attack] - [View IDS log]

Its same as low level. Take look how the page work and the inject malicious code for check the vulnerability.

Code: [<script>alert("this is medium level test !")</script>] this time not working its do what its mean to be:



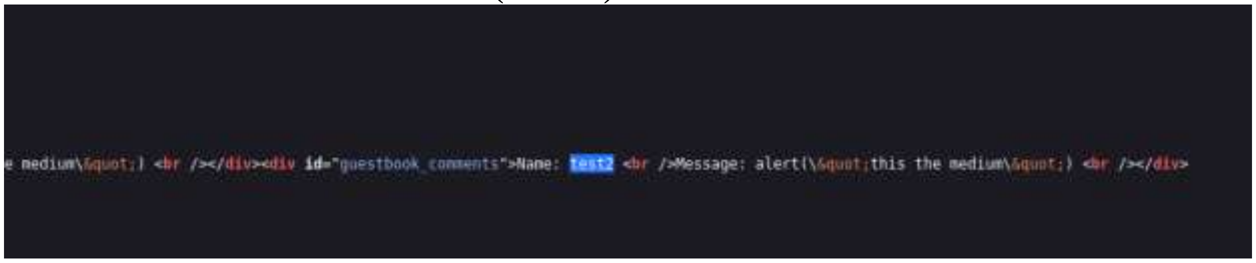
SQL Injection (Blind)
Upload
XSS reflected
XSS stored

Message: This is a test comment.

Name: test2
Message: alert(\"this the medium\")

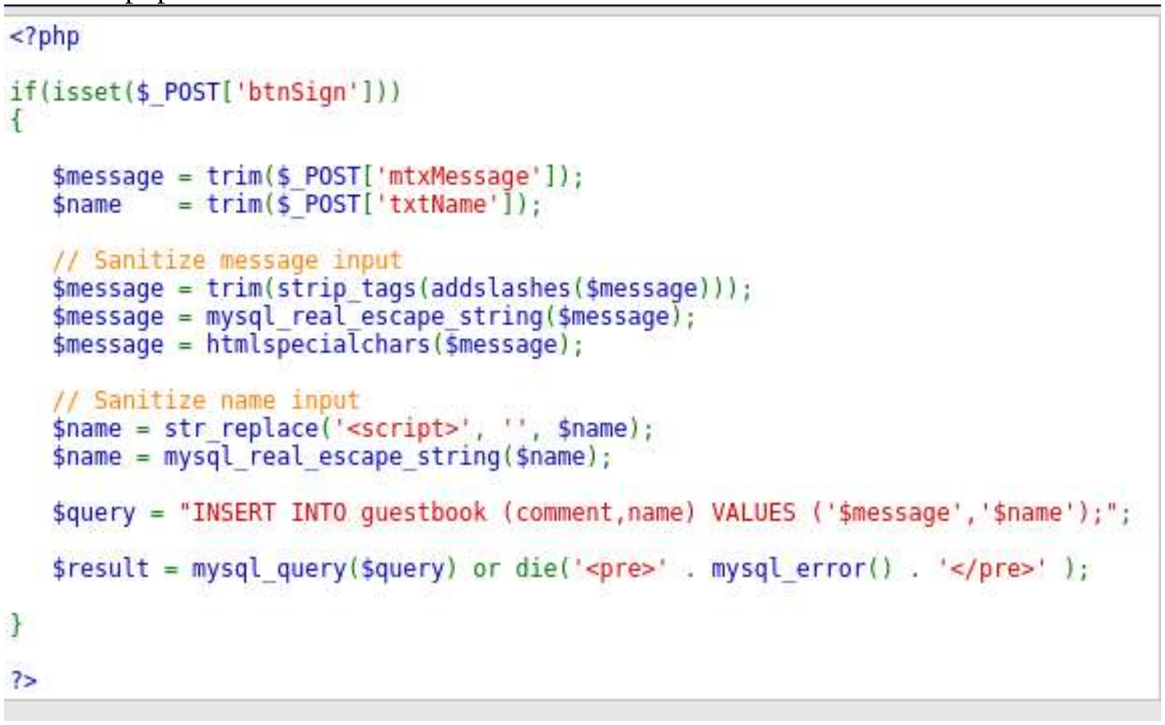
More info

Look for backend code/ source code : (Ctrl + u)



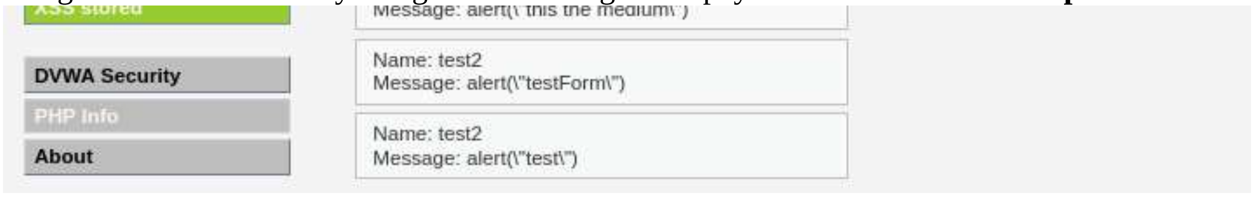
```
e medium\&quot;) <br /></div><div id="guestbook_comments">Name: test2 <br />Message: alert(\&quot;this the medium\&quot;) <br /></div>
```

Look for php code :



```
<?php

if(isset($_POST['btnSign']))
{

   $message = trim($_POST['mtxMessage']);
   $name    = trim($_POST['txtName']);

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

   // Sanitize name input
   $name = str_replace('<script>', '', $name);
   $name = mysql_real_escape_string($name);

   $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name');";

   $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );

}

?>
```

In this code seems my <script> tag vanished by [**`$name = str_replace('<script>', '', $_GET[ 'name' ] );`**] , this piece of JavaScript code. 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>.**

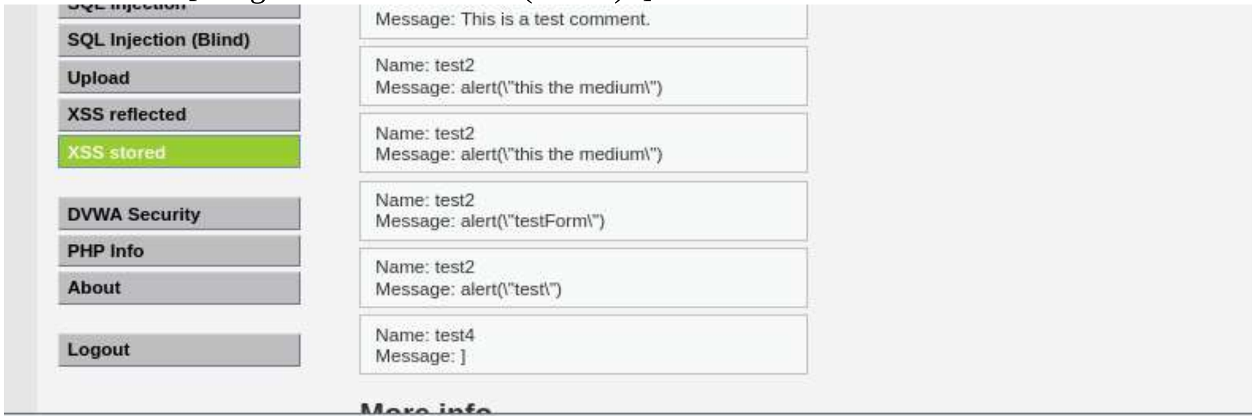


Message: alert(\"this the medium\")

Name: test2
Message: alert(\"testForm\")

Name: test2
Message: alert(\"test\")

This also not working I thing php code vanished script tag. So now I can use another tag to see is it work or not.[<img src=x onerror=alert("test!")>]



Message: This is a test comment.

Name: test2
Message: alert(\"this the medium\")

Name: test2
Message: alert(\"this the medium\")

Name: test2
Message: alert(\"testForm\")

Name: test2
Message: alert(\"test\")

Name: test4
Message:]

This also not working as php code sanitize all the html tag by make them not working.

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.

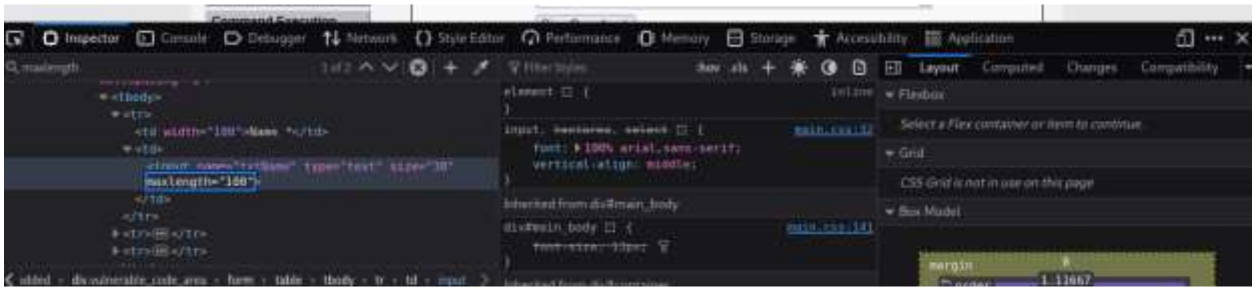
We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags.

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.

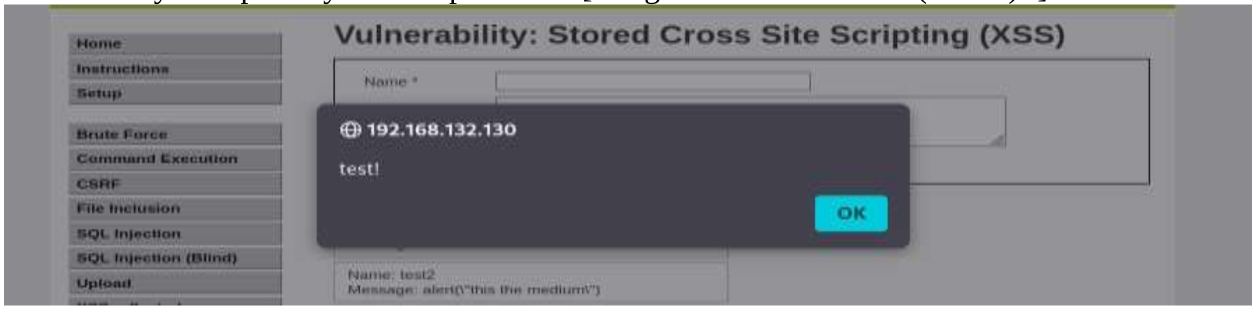
With the **maxlength** restriction changed, the XSS payload can now be entered into the **Name *** field.

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



```
▾ <tbody>
  ▾ <tr>
      <td width="100">Name *</td>
    ▾ <td>
        <input name="txtName" type="text" size="30"
        maxlength="100">
      </td>
    </tr>
  ▸ <tr>⊞</tr>
  ▸ <tr>⊞</tr>
```

This time try to exploit by name input filed: .[<img src=x onerror=alert("test!")>]



Vulnerability: Stored Cross Site Scripting (XSS)

192.168.132.130

test!

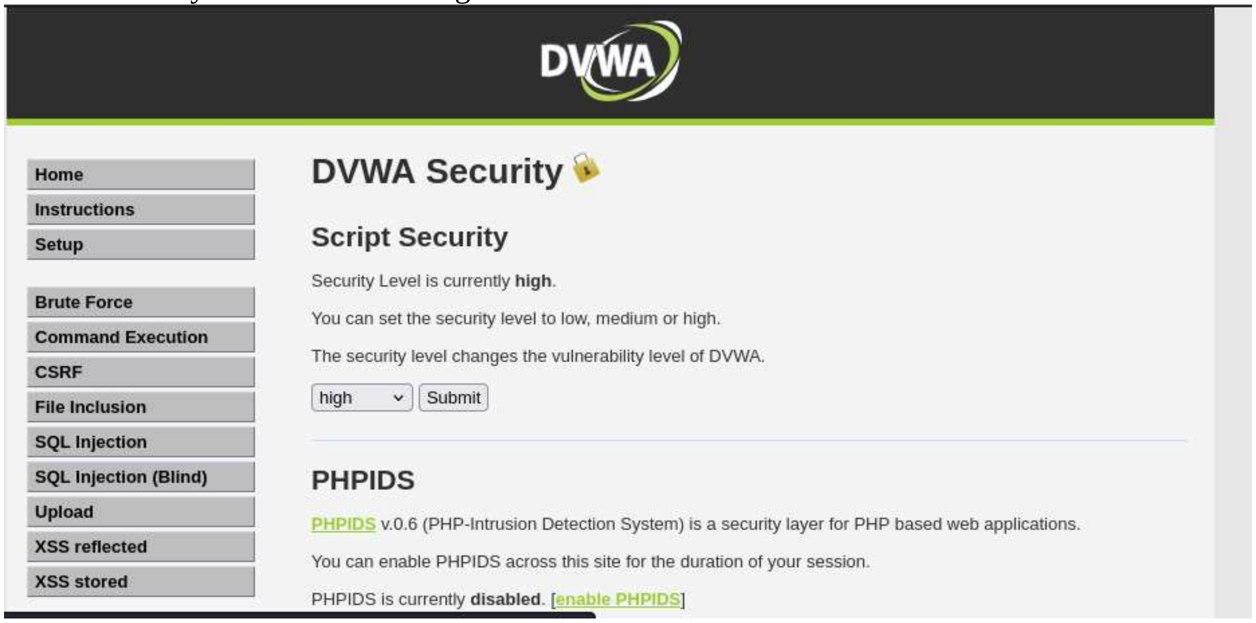
Name: test2
Message: alert(\"this the medium\")

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 successfully exploited Stored XSS vulnerability at the Medium level of security.

High security XSS Stored Attack

Submit security level medium to high :



DVWA Security

Script Security

Security Level is currently **high**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

PHPIDS

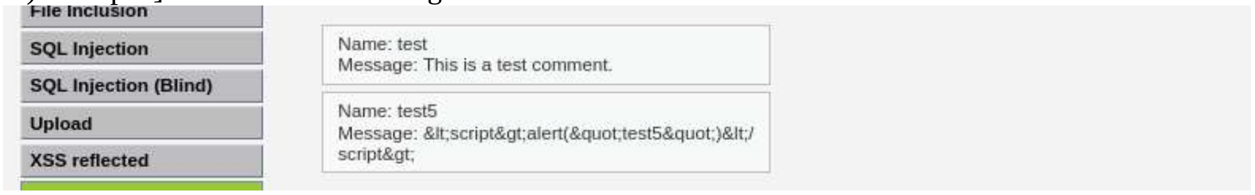
PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled**. [enable PHPIDS]

Its same as previous two level. Take look how the page work and the inject malicious code for check the vulnerability.

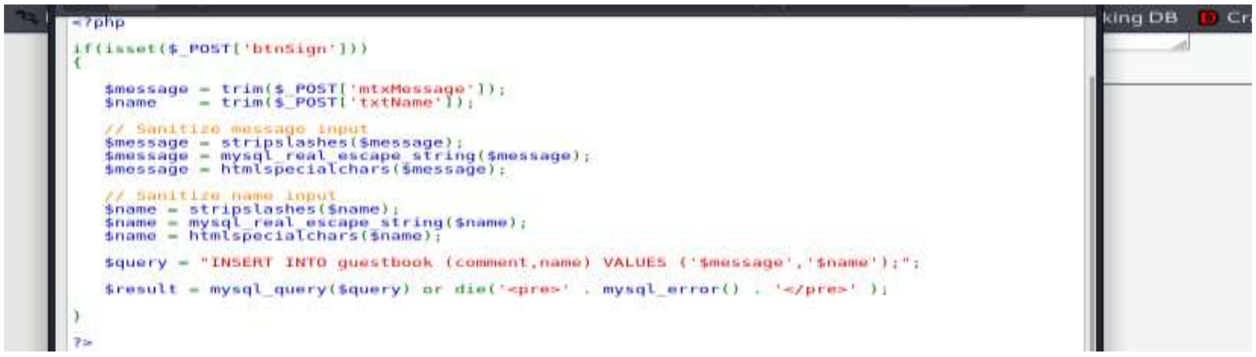
Code: [<script>alert("test5 !")</script>] and [<SCRIPT>alert("this is medium level test !")</script>] this time not working its do what its mean to be:



Name: test
Message: This is a test comment.

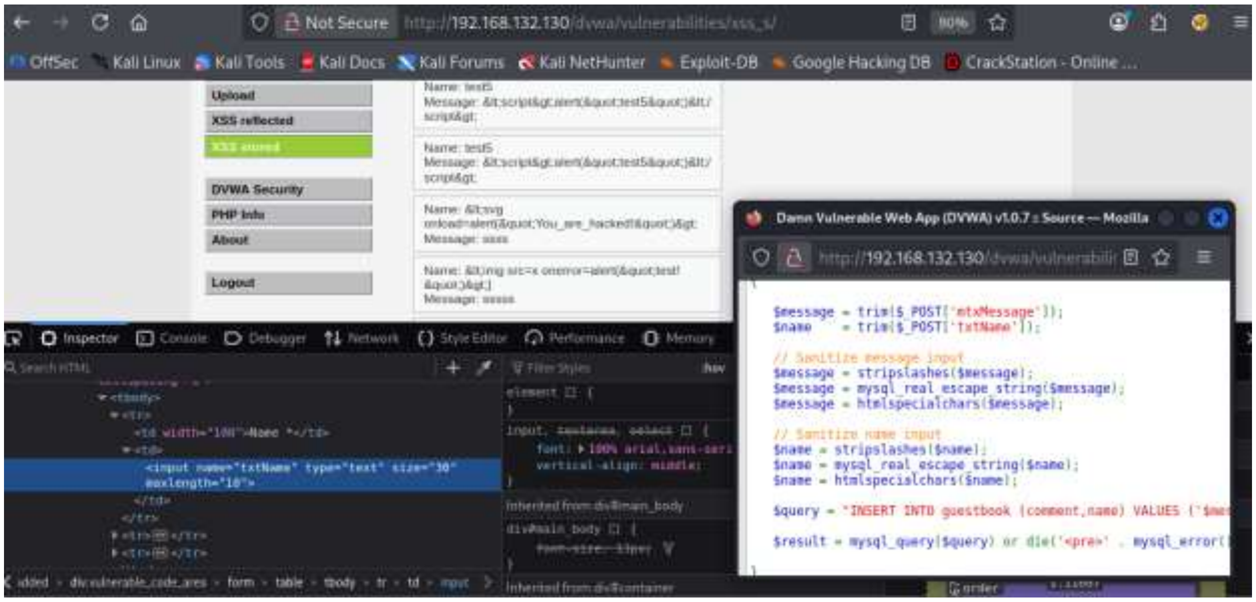
Name: test5
Message: &lt;script>alert("test5")</script>

Lets take look its source code :



```
<?php

if(isset($_POST['btnSign']))
{

   $message = trim($_POST['mtxMessage']);
   $name    = trim($_POST['txtName']);

   // Sanitize message input
   $message = stripslashes($message);
   $message = mysql_real_escape_string($message);
   $message = htmlspecialchars($message);

   // Sanitize name input
   $name = stripslashes($name);
   $name = mysql_real_escape_string($name);
   $name = htmlspecialchars($name);

   $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name');";

   $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );

}

?>
```

Its validetion code (!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == '')amd(echo 'Hello ' . htmlspecialchars($_GET['name']);)meke all the input value as a string and make all the specialcharecter like alternete . so, those Not respond like tag anymore . so malicious code didn't work as before low and medium levels.


Name: test5
Message: &lt;script&gt;alert(&quot;test5&quot;)&lt;/script&gt;

Name: test5
Message: &lt;script&gt;alert(&quot;test5&quot;)&lt;/script&gt;

Name: &lt;svg onload=alert(&quot;You_are_hacked!&quot;)&gt;
Message: ssss

Name: &lt;img src=x onerror=alert(&quot;test!&quot;)&gt;]
Message: sssss

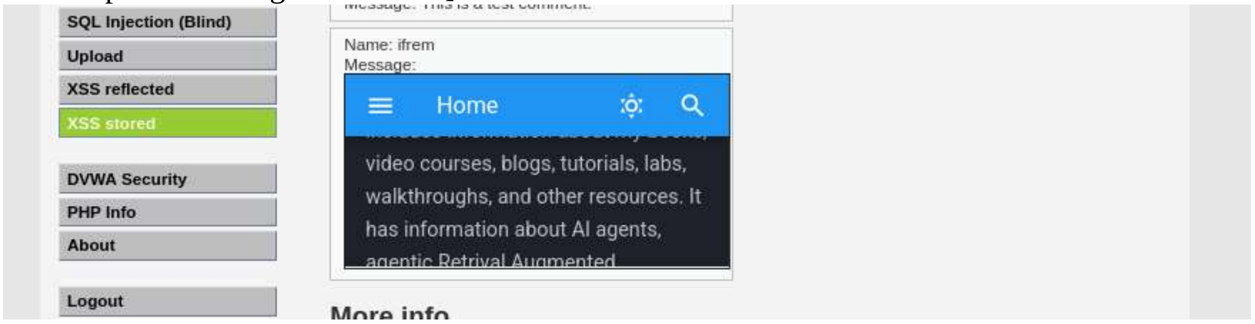
```
$message = trim($_POST['mtxMessage']);
$name    = trim($_POST['txtName']);

// Sanitize message input
$message = stripslashes($message);
$message = mysql_real_escape_string($message);
$message = htmlspecialchars($message);

// Sanitize name input
$name = stripslashes($name);
$name = mysql_real_escape_string($name);
$name = htmlspecialchars($name);

$query = "INSERT INTO guestbook (comment,name) VALUES ('$mes

$result = mysql_query($query) or die('<pre>' . mysql_error(
```

Its seems that this high level page have no vulnerability of XSS Stored. So it's cannot be compromised by XSS Stored attack.

Stored iframe exploit

Lets test Stored iframe exploit in low security DVWA

1st set security level low and select XSS Stored . 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>]



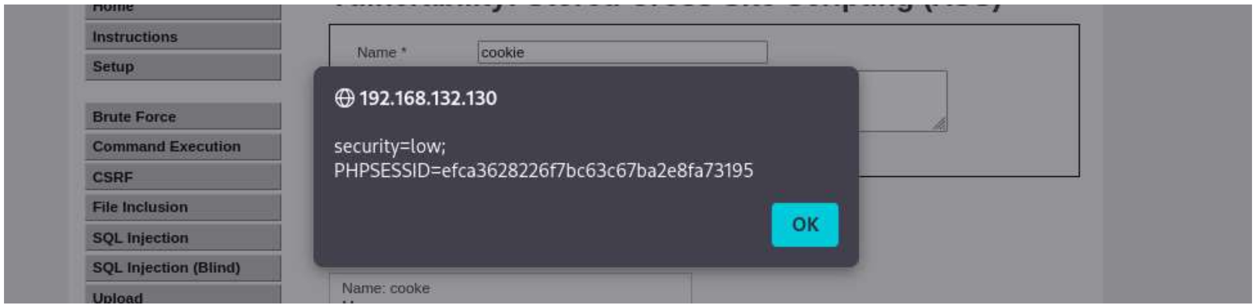
Name: ifrem
Message:

video courses, blogs, tutorials, labs, walkthroughs, and other resources. It has information about AI agents,

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

Stored cookie exploit

In XSS Stored in the name filed use cookie and massage filed input load a payload : [<script>alert(“document.cookie”)</script>] for get php session cookie. This is a cookie that PHP uses to keep of track of running sessions.



cookie

192.168.132.130

```
security=low;
PHPSESSID=efca3628226f7bc63c67ba2e8fa73195
```

Name: cooke

So this is exploit by different kind of payload. Mean when a web application is found vulnerable to XSS at low, medium, or high levels during an ethical hacking test, it indicates the maturity of its security controls. A low-level XSS vulnerability shows that the application has little or no input validation or output encoding, making it highly insecure and easy to exploit. A medium-level vulnerability suggests that some security measures are in place, such as basic filtering or blacklisting, but these controls are weak and can be bypassed by attackers. A high-level XSS vulnerability means the application has stronger and more thoughtful security mechanisms, yet still fails to handle certain advanced or context-specific attack scenarios. Overall, the higher the level at which XSS is still possible, the more security awareness exists, but it also highlights gaps that must be addressed to achieve secure coding standards.

------------------End---------------