



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/>

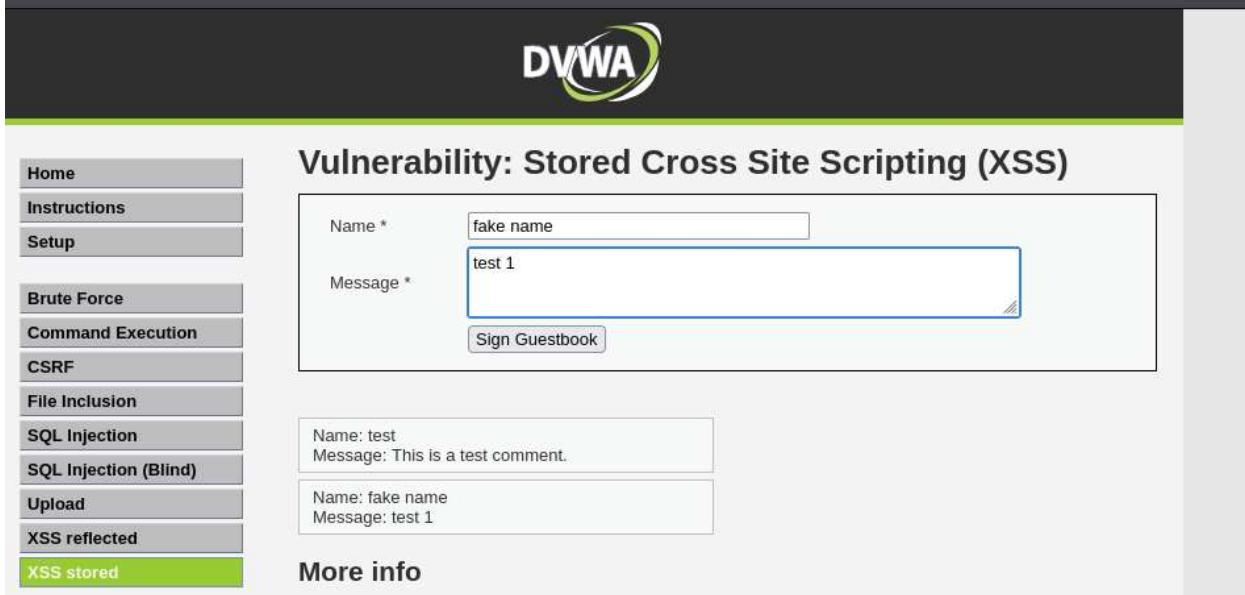
GitHub: <https://github.com/jobarulislam>

Phone: +880 1312-678017

Login to the dvwa and select XSS reflected :

Low security XSS Stored Attack

1st Observe how its working :



The screenshot shows the DVWA interface with the 'XSS reflected' menu item selected. The main content area displays a guestbook form and its output. The form fields are:

- Name *: fake name
- Message *: test 1

Below the form, the output shows two entries:

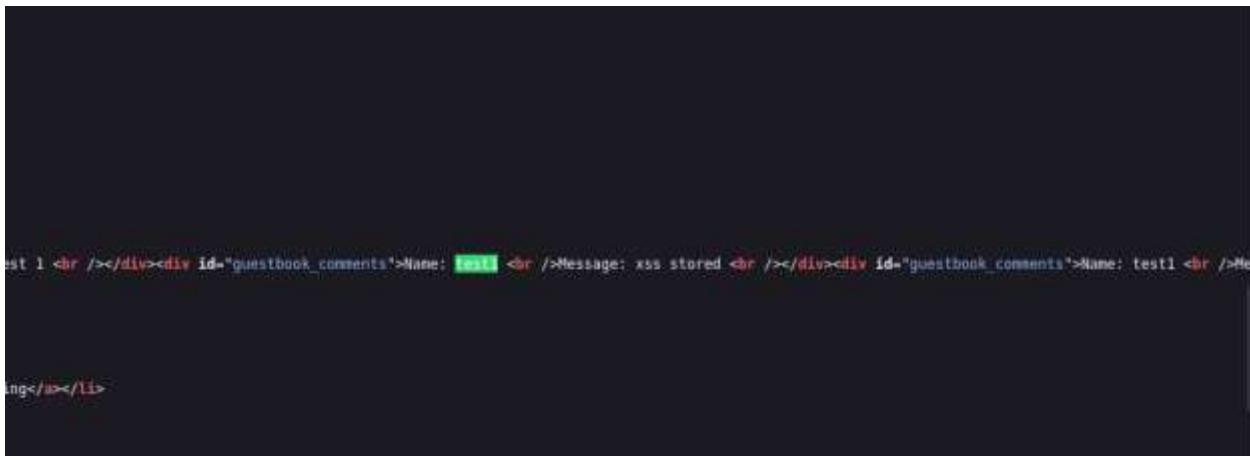
- Name: test
Message: This is a test comment.
- Name: fake name
Message: test 1

A 'More info' link is visible at the bottom of the content area.

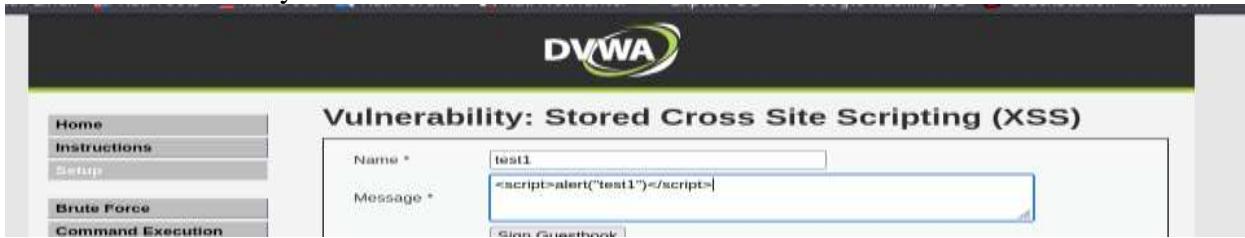
Use (Ctrl + u) for open source code:

(Ctrl + f) for search [hello reflected_text]

Back to Render window by (Ctrl + w)



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



The screenshot shows the DVWA interface with the 'XSS reflected' menu item selected. The main content area displays a guestbook form. The form fields are:

- Name *: test1
- Message *: <script>alert("test1")</script>

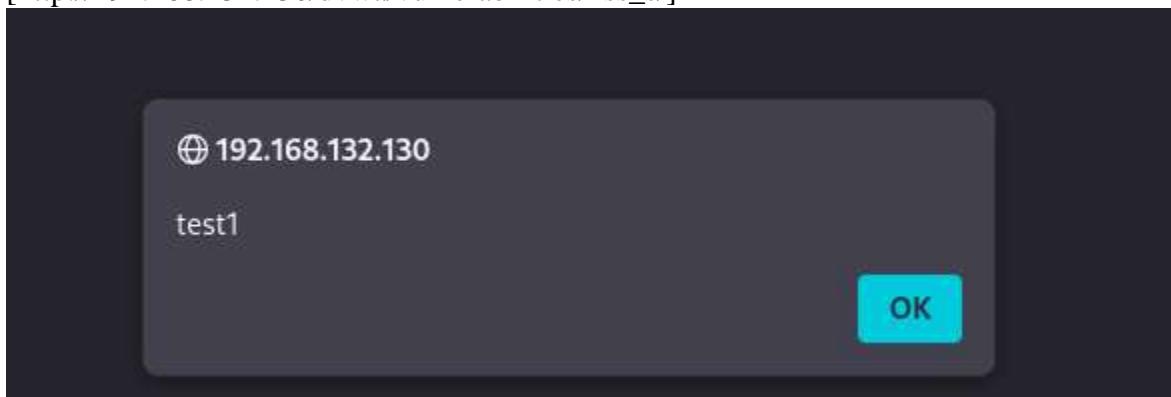
The 'Sign Guestbook' button is visible below the message field.

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

A screenshot of a web browser displaying the DVWA (Damn Vulnerable Web Application) interface. The title bar shows the URL: http://192.168.132.130/dvwa/vulnerabilities/xss_s/. The main content area is titled "Vulnerability: Stored Cross Site Scripting (XSS)". On the left, there's a sidebar with links like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, and SQL Injection. In the center, there's a modal dialog box with the IP address "192.168.132.130" and the text "test1". Below the dialog, a status message says "Message: This is a test command".

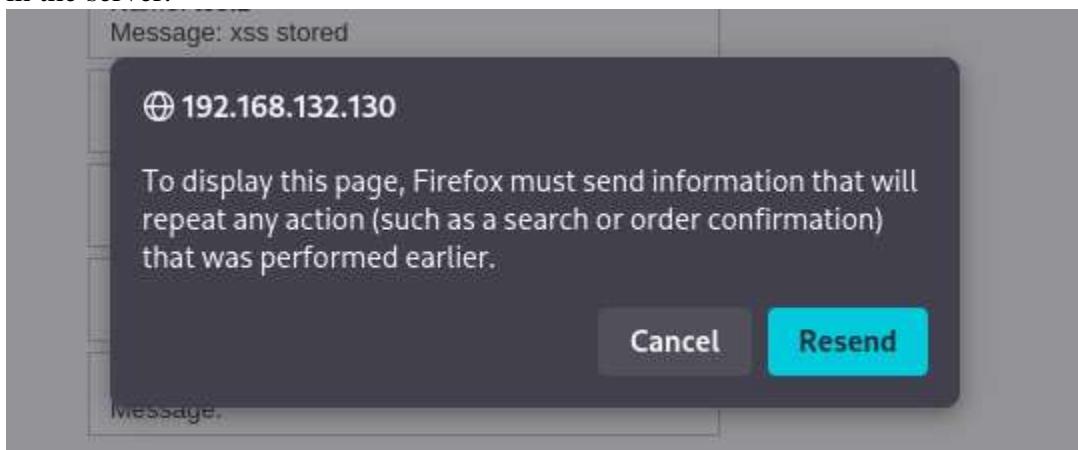
Url of this page :

[http://192.168.132.130/dvwa/vulnerabilities/xss_s/]



Its seems that this page have vulnerability and I found it. So it's can be 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.



Medium security XSS Stored Attack

Submit security level low to medium:

The screenshot shows the DVWA Security interface. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The 'XSS stored' option is highlighted with a green background. The main content area is titled 'Script Security' and displays the message 'Security Level is currently **low**'. It says you can set the security level to low, medium or high, and notes that the security level changes the vulnerability level of DVWA. A dropdown menu shows 'medium' selected, and a 'Submit' button is present. Below this is a section titled 'PHPIDS' with the subtext: '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.' It also mentions that PHPIDS is currently disabled and provides a link to enable it. At the bottom of the main content area are links for '[Simulate attack]' and '[View IDS log]'. The top navigation bar has a 'DVWA Security' logo.

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:

The screenshot shows the DVWA XSS stored attack results. The left sidebar still has the 'XSS stored' option highlighted. The main content area shows two input fields: 'Message' containing 'Message: This is a test comment.' and 'Name' containing 'Name: test2'. Below these fields is a text area labeled 'Message' containing the injected code: 'Message: alert(\"this is the medium\")'. To the right of this text area is a 'More info' button.

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

The screenshot shows the DVWA source code. The injected code 'Message: alert("this is the medium")' is visible in the browser's developer tools under the 'Elements' tab, specifically within the 'guestbook_comments' div. The code is highlighted in red, indicating it is a script tag.

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>.

The screenshot shows the DVWA XSS stored page. On the left, there's a sidebar with links: DVWA Security, PHP Info, About, and Logout. The main area has two entries in a table:

XSS stored	Message: alert('this the medium')
DVWA Security	Name: test2 Message: alert("testForm")
PHP Info	
About	

This also not working I think php code vanished script tag. So now I can use another tag to see is it work or not.[]

The screenshot shows the DVWA XSS stored page. The sidebar and table structure are identical to the previous screenshot, but the entries show various tags used in the payload:

XSS stored	Message: This is a test comment.
SQL Injection (Blind)	
Upload	Name: test2 Message: alert("this the medium")
XSS reflected	Name: test2 Message: alert('this the medium')
XSS stored	Name: test2 Message: alert("testForm")
DVWA Security	Name: test2 Message: alert("test")
PHP Info	
About	

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.

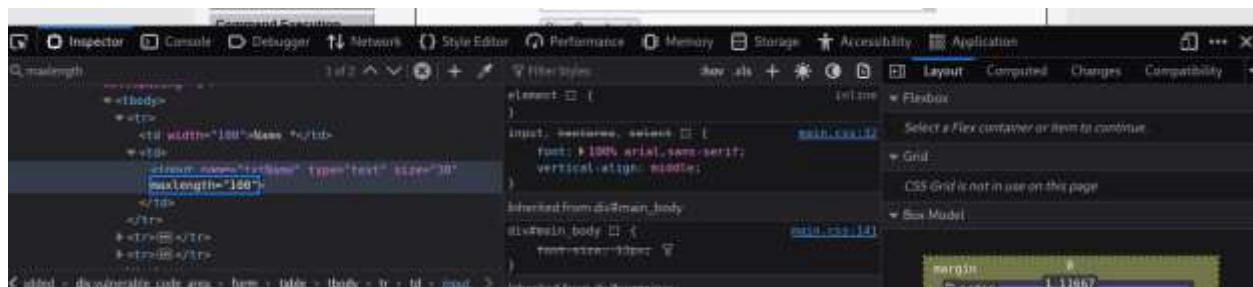
We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags.

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.]



This time try to exploit by name input filed: .[]

A screenshot of a web application titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. On the left, there is a sidebar with various menu items like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, and others. The main content area has a form with a 'Name *' input field containing 'test!'. Below the input field, a message box displays 'Message: alert('this the medium')'. At the bottom of the message box, there is an 'OK' button.

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 :

The screenshot shows the DVWA Security interface. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected (selected), and XSS stored. The main content area is titled "DVWA Security" with a lock icon. It displays the "Script Security" section, which states "Security Level is currently **high**". Below this, it says "You can set the security level to low, medium or high." and "The security level changes the vulnerability level of DVWA." A dropdown menu is set to "high" and a "Submit" button is present. Further down, the "PHPIDS" section is shown, stating "PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications." It says "You can enable PHPIDS across this site for the duration of your session." and "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:

The screenshot shows the DVWA File Inclusion page. The sidebar menu includes File Inclusion (selected), SQL Injection, SQL Injection (Blind), Upload, and XSS reflected. The main content area has two text input fields. The top field contains "Name: test" and "Message: This is a test comment.". The bottom field contains "Name: test5" and "Message: <script&gtalert("test5");</script&gt".

Lets take look its source code :

The screenshot shows a code editor window displaying PHP source code for a guestbook script. The code includes logic to handle POST requests for signing in, sanitize inputs for message and name, and insert them into a database. The key part of the code related to XSS is the insertion query:

```
<?php
if(isset($_POST['btnSign'])) {
    $message = trim($_POST['txtMessage']);
    $name = trim($_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 validation code (!array_key_exists ("name", \$_GET) || \$_GET['name'] == NULL || \$_GET['name'] == '')amd(echo 'Hello ' . htmlspecialchars(\$_GET['name']));)make all the input value as a string and make all the specialcharacter like alternete . so, those Not respond like tag anymore . so malicious code didn't work as before low and medium levels.

The screenshot shows the DVWA interface with the 'XSS stored' tab selected. Below it, four messages are listed:

- Name: test
Message: <script>alert('test')</script>
- Name: test5
Message: <script>alert('test5')</script>
- Name: attack
Message:
- Name: attack2
Message: aaaaa

On the right, a Mozilla Firefox developer tools window shows the source code of the guestbook insertion script:

```

$message = trim($_POST['txMessage']);
$name = trim($_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());

```

It 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>]

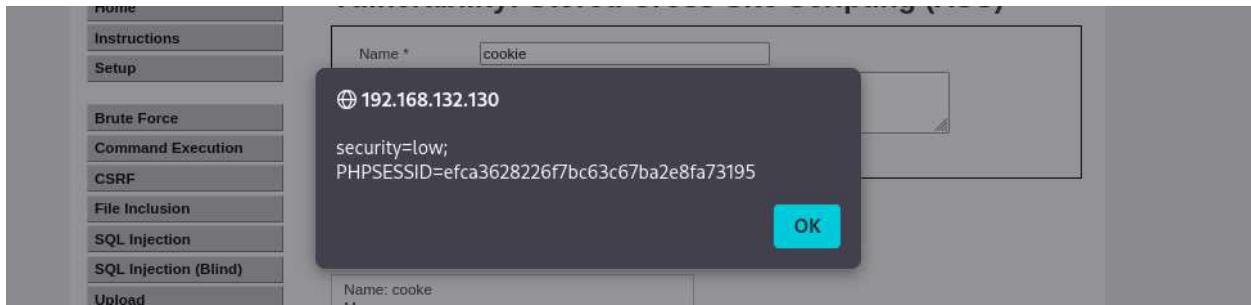
The screenshot shows the DVWA interface with the 'XSS stored' tab selected. The 'Name' field contains 'ifrem' and the 'Message' field contains the message from the previous step. The resulting guestbook entry is displayed:

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

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.



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-----