Lab - Cross Site Scripting

Nama : Fitri Sri Wahyuni

Nim : 2211082036

Kelas : TRPL 3B

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.

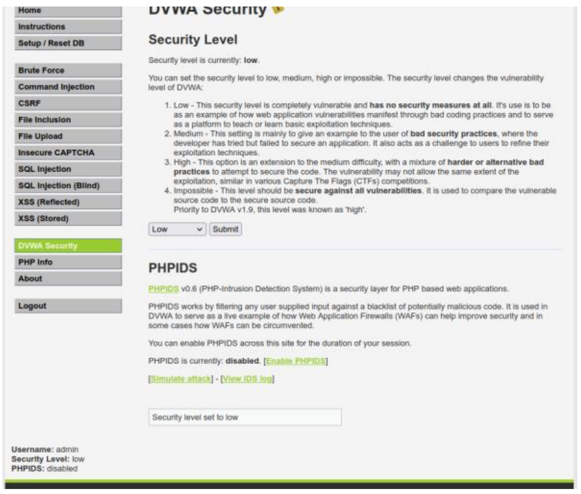
a. From the Kali Linux VM, open a browser and navigate to the DVWA application.

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**.



DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

Low Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

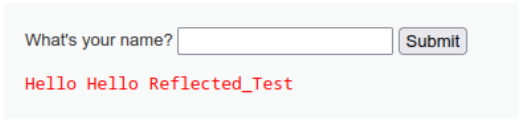
Security level set to low

Username: admin
Security Level: low
PHPIDS: disabled

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.



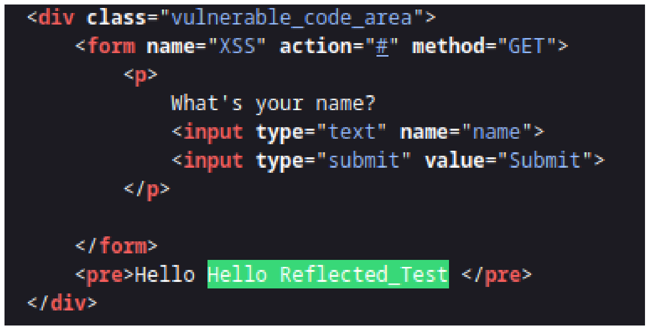
What's your name? Submit

```
Hello Hello Reflected_Test
```

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



```
<div class="vulnerable_code_area">
    <form name="XSS" action="#" method="GET">
        <p>
            What's your name?
            <input type="text" name="name">
            <input type="submit" value="Submit">
        </p>

    </form>
    <pre>Hello Hello Reflected_Test </pre>
</div>
```

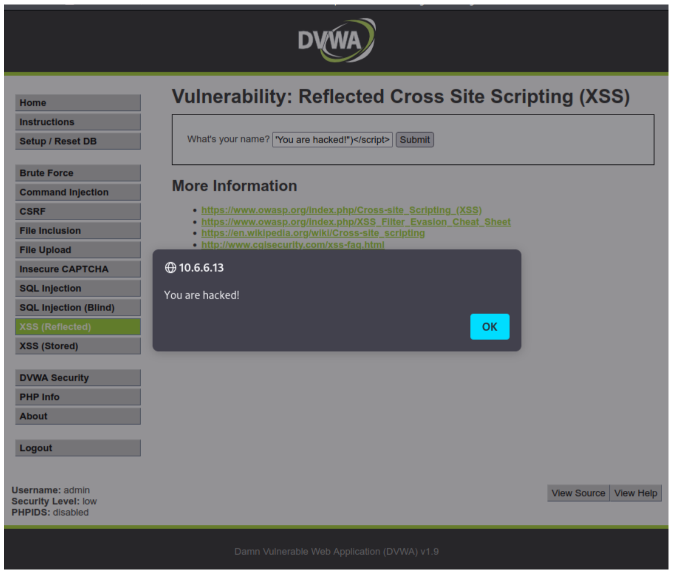
.

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.



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? "You are hacked!")</script>

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html

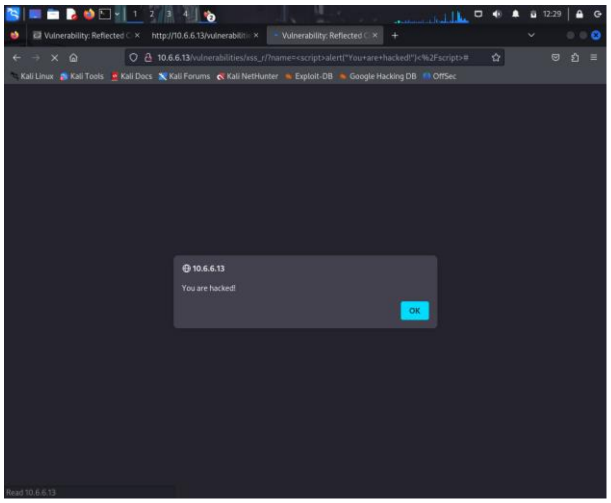
10.6.6.13

You are hacked!

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.9

h. Select and copy the URL for the compromised page. Open a new browser tab and paste the URL into the URL field and press <Enter>.



10.6.6.13/vulnerabilities/xss_r/?name=<script>alert("You+are+hacked!")<%2Fscript>#

10.6.6.13

You are hacked!

Read 10.6.6.13

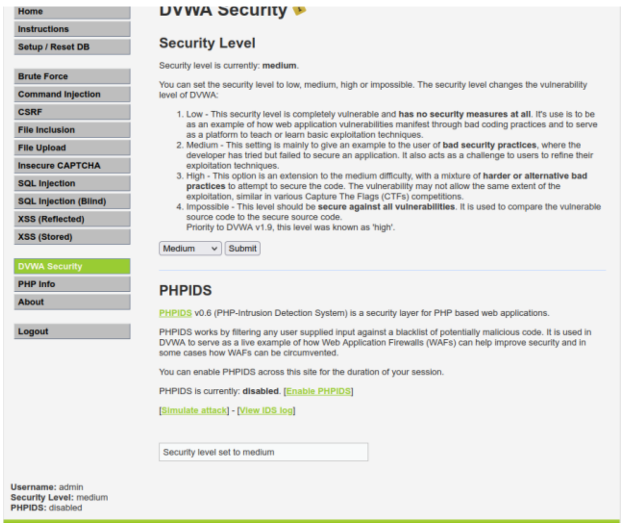
You should see the same web page appear displaying the **You are hacked**! popup box. This means that if a user opens the URL a malicious script will execute. The alert box is used to simulate a malicious script in this lab.

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**.



DVWA Security

Security Level

Security level is currently: **medium**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

Medium | Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

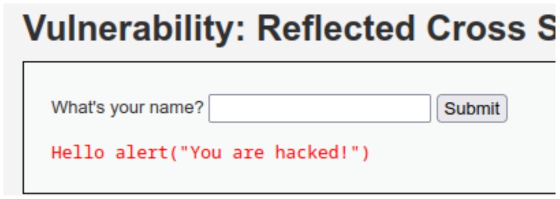
Security level set to medium

Username: admin
Security Level: medium
PHPIDS: disabled

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>
```



Vulnerability: Reflected Cross S

What's your name? Submit

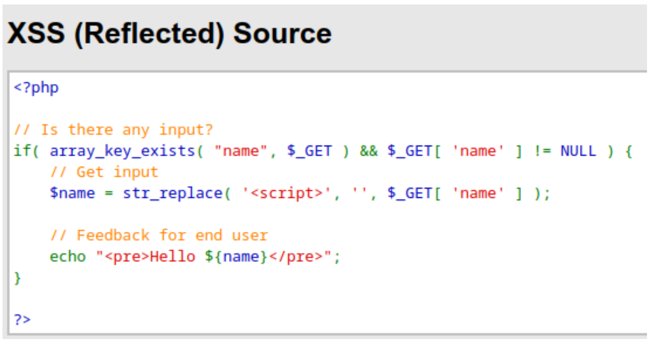
```
Hello alert("You are hacked!")
```

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.



XSS (Reflected) Source

```
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

e. Note the line:

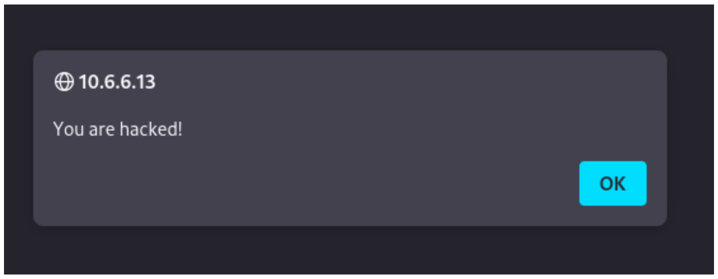
```
$name = str_replace ( '<script>', '',  $_GET[ 'name' ] );
```

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>
```



10.6.6.13

You are hacked!

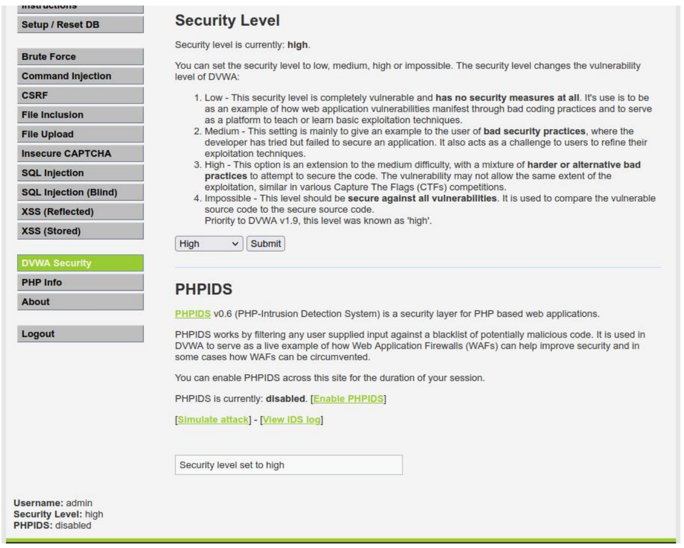
Did the popup alert appear? If so, why?

Ya, alert muncul karena filter hanya menghapus <script> huruf kecil saja. <ScRipt> berhasil melewati filter..

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**.



Security Level

Security level is currently: **high**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

High Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

Security level set to high

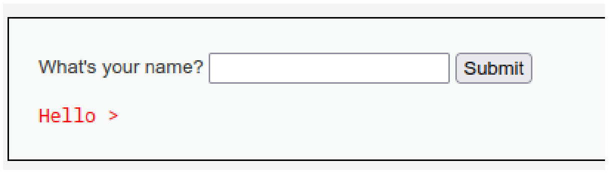
Username: admin
Security Level: high
PHPIDS: disabled

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.



What's your name?

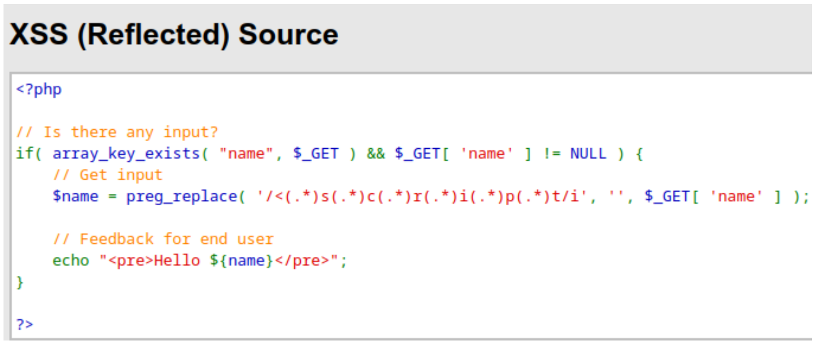
Hello >

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.



XSS (Reflected) Source

```
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

Which character in the script was omitted from the regular expression? How do you know?

Tidak ada karakter yang dihilangkan. Regex menggunakan semua karakter dalam `script`, tapi kita tahu filter bekerja karena tidak ada alert muncul..

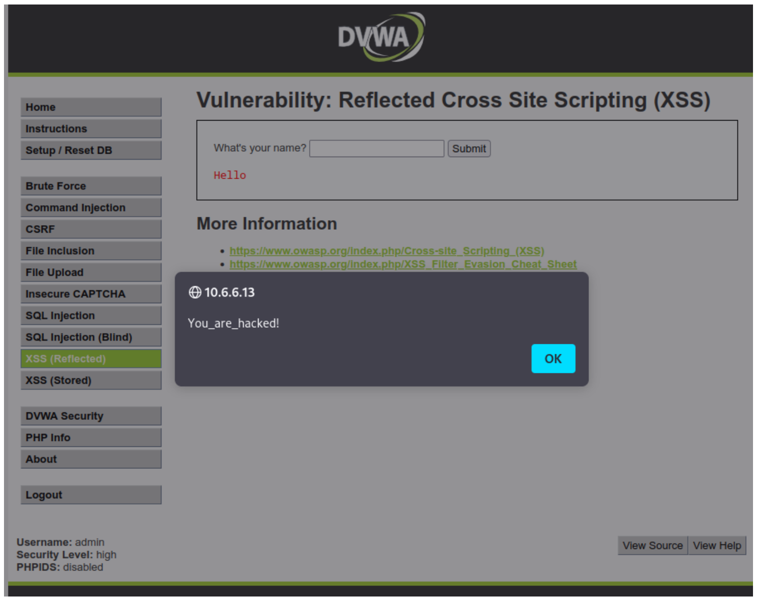
e. To bypass this filter, we must use another HTML tag instead of **<script>** to attack the site.

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.



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

10.6.6.13

You_are_hacked!

Username: admin
Security Level: high
PHPIDS: disabled

Review the text that you input into the web form. How did it work?

Karena tag `<img>` dengan `onerror` tidak difilter oleh backend, JavaScript berhasil dijalankan saat gambar gagal dimuat..

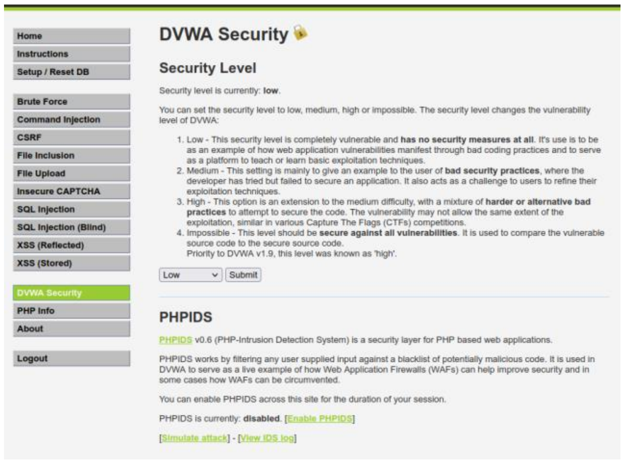
Part 2: Perform Stored Cross Site Scripting Exploits

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**.



DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
Priority to DVWA v1.9, this level was known as 'high'.

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

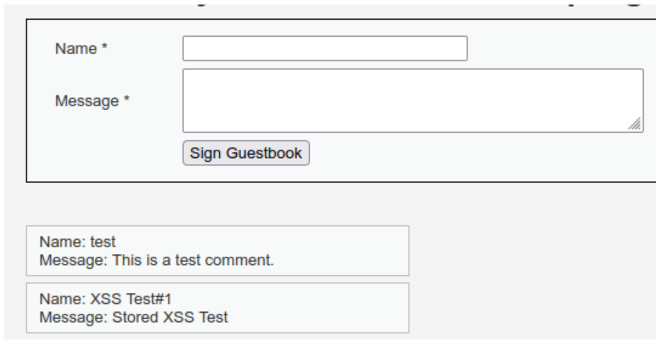
You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

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**.

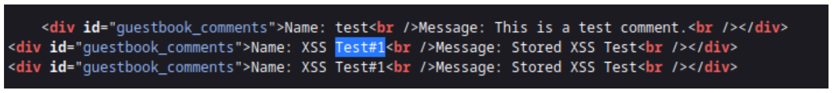


Name: test
Message: This is a test comment.

Name: XSS Test#1
Message: Stored XSS Test

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.



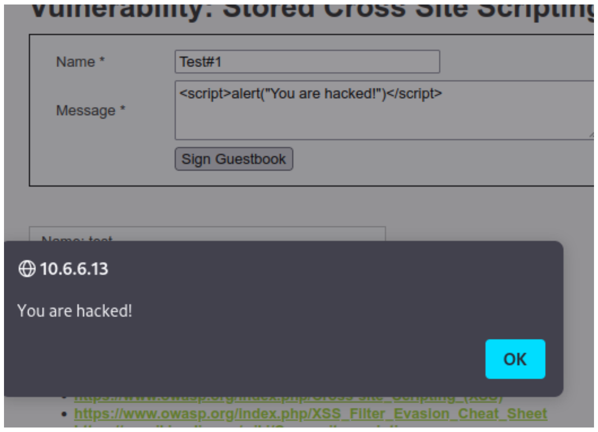
```
    <div id="guestbook_comments">Name: test<br />Message: This is a test comment.<br /></div>
<div id="guestbook_comments">Name: XSS Test#1<br />Message: Stored XSS Test<br /></div>
<div id="guestbook_comments">Name: XSS Test#1<br />Message: Stored XSS Test<br /></div>
```

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.



Name *: Test#1

Message *: <script>alert("You are hacked!")</script>

10.6.6.13

You are hacked!

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

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.

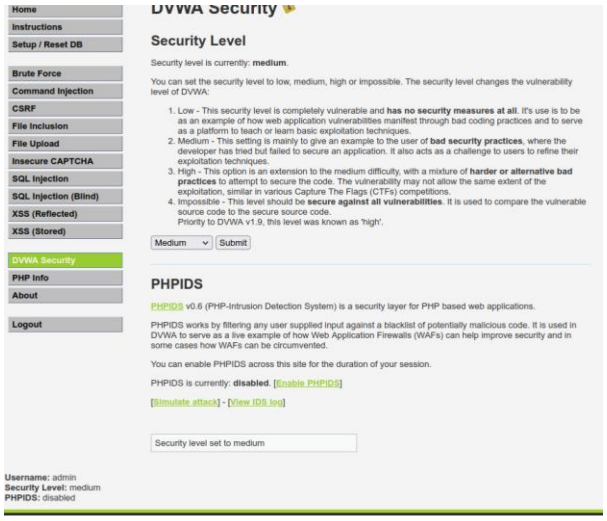
h. Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**.

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**.



DVWA Security

Security Level

Security level is currently: **medium**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

Medium

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

Security level set to medium

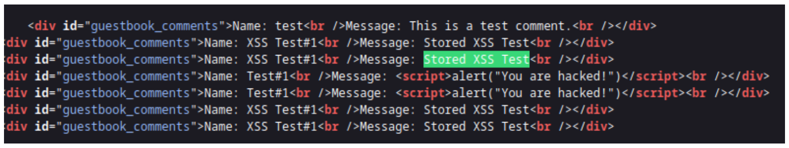
Username: admin
Security Level: medium
PHPIDS: disabled

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.



```
  <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>
div id="guestbook_comments">Name: Test#1<br />Message: <script>alert("You are hacked!")</script><br /></div>
div id="guestbook_comments">Name: Test#1<br />Message: <script>alert("You are hacked!")</script><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>
```

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.

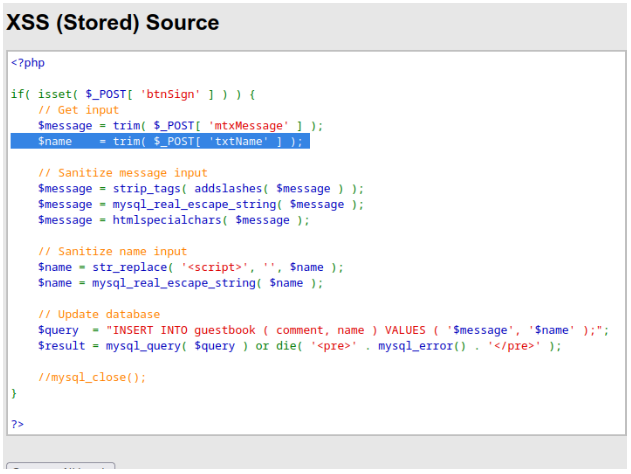
This means that there is code in the backend that is sanitizing the user input from the **Message *** field to prevent scripts from being submitted. You can see the modified input in the last rectangle message box below the input fields.

How did the input filter script modify the input?

`strip_tags()` menghapus HTML tags, `htmlspecialchars()` mengubah karakter khusus, jadi payload jadi teks biasa, bukan kode..

g. 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.



XSS (Stored) Source

```
<?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 = str_replace( '<script>', '', $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();
}

?>
```

Research the PHP addslashes() function on the internet. How did it change the input?

Fungsi `addslashes()` menambahkan backslash (\) sebelum tanda kutip (', ") untuk mencegah injeksi kode..

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.

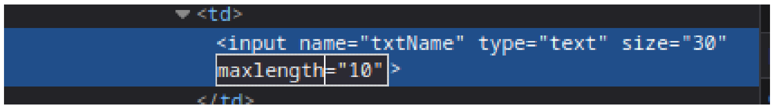
We can attempt to bypass the security on the **Name *** field by using some other payload that does not contain **<script>** tags.

h. Close the source code window.

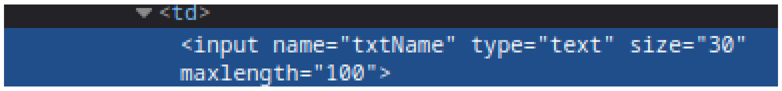
i. Before entering any payload into the **Name *** field, the max character length restriction of 10 characters on the field must be increased. This is a client-side setting so it is easy to change with the following steps:

   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.



```
<td>
  <input name="txtName" type="text" size="30"
  maxlength="10">
</td>
```



```
<td>
  <input name="txtName" type="text" size="30"
  maxlength="100">
```

   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.

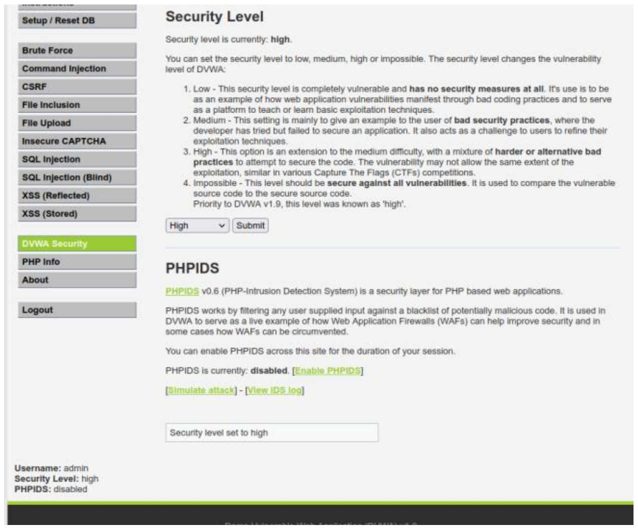
l. Before proceeding to the next step, clear the XSS payload from the page. Click **Setup / Reset DB** in the left menu then click **Create / Reset Database**.

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**.



Security Level

Security level is currently: **high**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

High Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

PHPIDS is currently: **disabled**. [Enable PHPIDS]

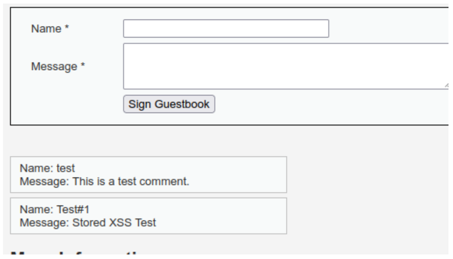
[Simulate attack] - [View IDS log]

Security level set to high

Username: admin
Security Level: high
PHPIDS: disabled

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**.



Name: test
Message: This is a test comment.

Name: Test#1
Message: Stored XSS Test

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>
```

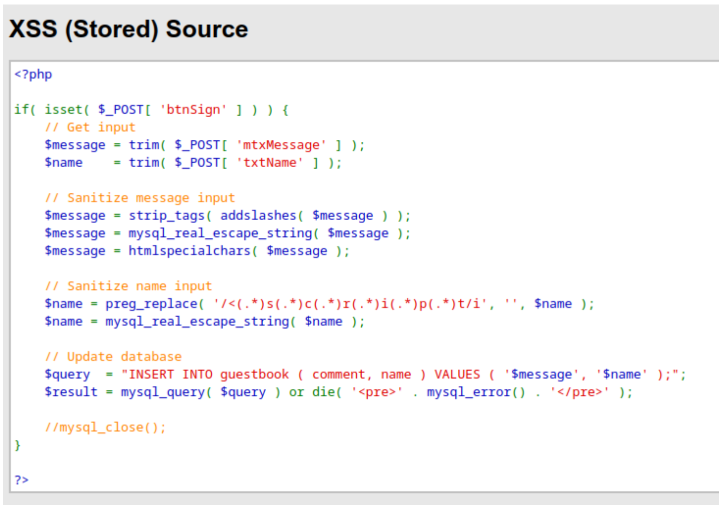
No popup box will appear. Refreshing the page will not cause the alert popup box to appear either.

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.



XSS (Stored) Source

```
<?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( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/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();
}

?>
```

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

h. 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.

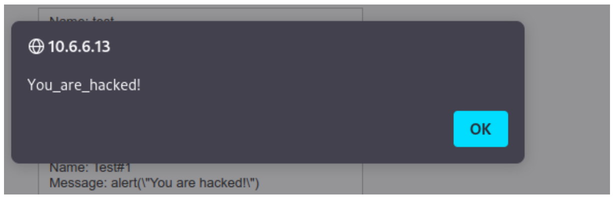
i. Return to the Vulnerability page and enter the following payload in the **Name *** field. (Note the use of underscores to replace spaces.)

```
<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.



10.6.6.13

You_are_hacked!

Name: Test#1
Message: alert(\"You are hacked!\")

   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**.

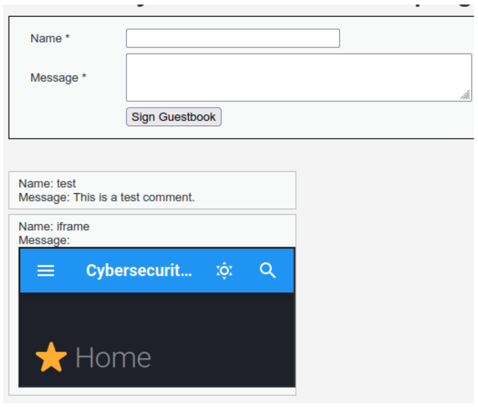
b. Select **XSS (Stored)** in the left menu.

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.



Name: test
Message: This is a test comment.

Name: iframe
Message:

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.

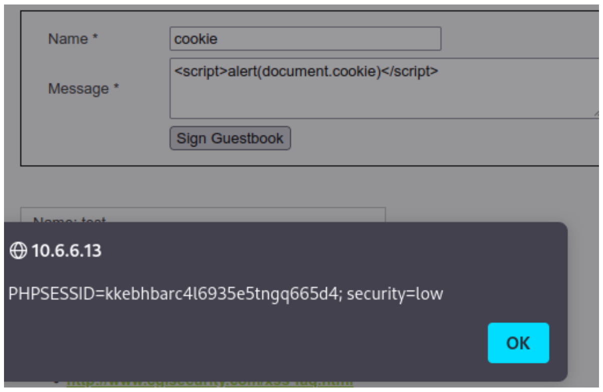
a. Select **DVWA Security** in the left menu and select **Low** in the Security Level options dropdown. Click **Submit**.

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.



Name * cookie

Message * <script>alert(document.cookie)</script>

10.6.6.13

PHPSESSID=kkebhbarc4l6935e5tngq665d4; security=low

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.

e. Try to steal cookies at the medium and high security levels using techniques that you have learned in this lab.

Reflection Question

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?

Kalau sebuah aplikasi masih bisa diserang XSS di level Low, itu artinya tidak ada perlindungan sama sekali terhadap input dari pengguna. Saat diserang di level Medium, artinya sudah ada upaya untuk memfilter input, tapi perlindungannya masih lemah dan bisa dilewati dengan trik sederhana. Kalau di level High masih bisa dibobol, berarti meskipun sudah ada filter dan sanitasi yang lebih ketat, hacker yang lebih berpengalaman masih bisa mencari celah untuk menyusup. Ini menunjukkan bahwa sistem keamanannya belum cukup kuat dan perlu diperbaiki agar tidak mudah diserang dan membahayakan data pengguna.