# Lab 16: Stored XSS

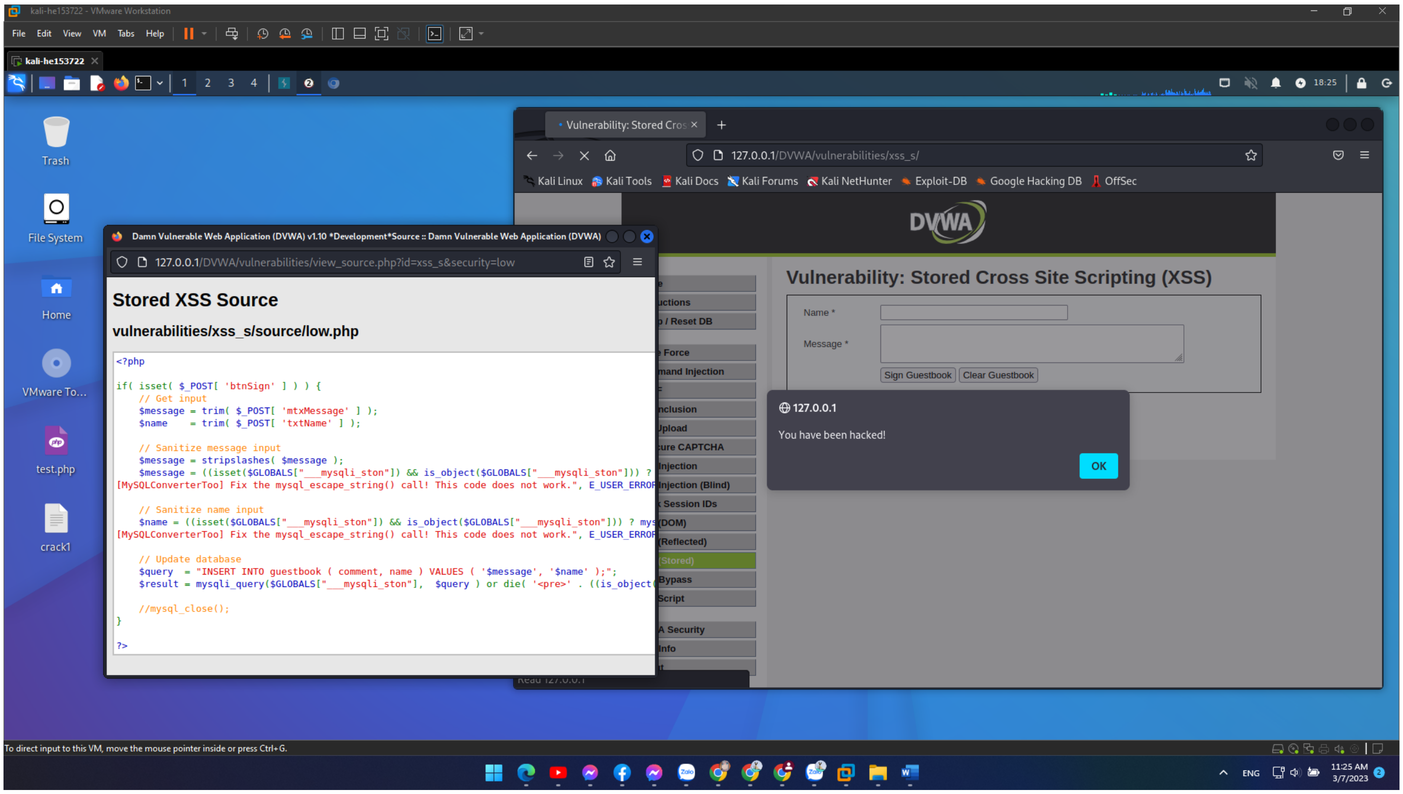
## Explain about Stored XSS? What is the difference between XSS techniques we learnt?

Stored XSS (Cross-Site Scripting) is a type of web application vulnerability where an attacker injects malicious scripts or code into a web application, which is then stored and displayed to other users who access the same page. The attack occurs when the application does not properly validate or sanitize user input, allowing the attacker to inject and store malicious code that can be executed by other users' browsers. The key difference between Stored XSS and other types of XSS attacks, such as Reflected XSS and DOM-based XSS, is the way in which the malicious script is delivered to users. In Stored XSS, the attacker injects the malicious script directly into the application's database, which is then retrieved and displayed to all users who access the affected page. This makes Stored XSS particularly dangerous, as the attack is persistent and affects all users who access the affected page, not just the user who initiated the attack. In Reflected XSS, the attacker injects the malicious script into a URL or form input field, which is then reflected back to the user's browser in the application's response. This type of XSS attack is less dangerous than Stored XSS, as it requires the user to click on a link or submit a form in order to execute the malicious script. In DOM-based XSS, the attacker injects the malicious script into the Document Object Model (DOM) of the web page, which is then executed by the user's browser when certain events are triggered. This type of XSS attack is also less dangerous than Stored XSS, as it is limited to a specific user and does not affect other users who access the same page. Overall, Stored XSS is considered the most dangerous type of XSS attack, as it can have far-reaching consequences and is often difficult to detect and mitigate.

## LOW

There isn't any input checking step so we can do it easily with the basic payload:

```
<script>alert("You have been hacked!");</script>
```



**Stored XSS Source**

**vulnerabilities/xss_s/source/low.php**

```php
<?php

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

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ?
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR

    // Sanitize name input
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mys
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object(

    //mysql_close();
}

?>
```

Vulnerability: Stored Cross Site Scripting (XSS)

127.0.0.1

You have been hacked!

kali-he153722 - VMware Workstation
127.0.0.1/DVWA/vulnerabilities/xss_s/
127.0.0.1/DVWA/vulnerabilities/view_source.php?id=xss_s&security=low
Name *
Message *
Sign Guestbook
Clear Guestbook
OK

## MEDIUM

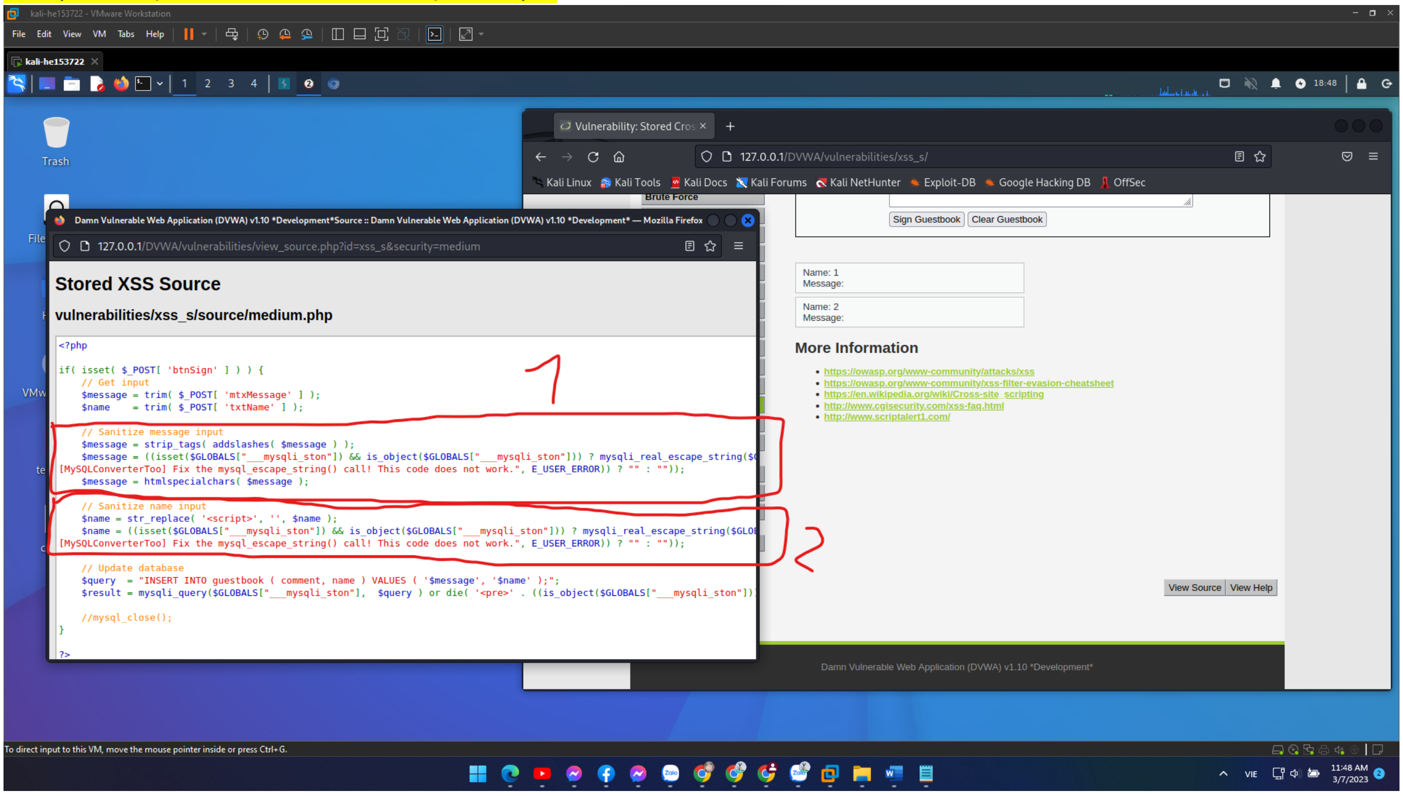
First block is for performing input sanitization on message field. This block contains two php functions for performing input sanitization. First one is strip_tags(). It removes all html tags from the message field before storing them in database. Second function is htmlspecialchars(). It converts all the bad characters like &, ", ', > and < in their equivalent HTML character encoding so they won't remain in their original form when they reflect back in the browser. You can confirm it by checking the page source.

So, in message field there are two levels of sanitization. Firstly strip_tags() function removes all the HTML tags from the message field and even if some text containing quotes or bad characters passes through this function, htmlspecialchars() will definitely encode them into equivalent HTML characters. So our XSS payload becomes useless. From here we conclude that the message field is completely secure and we cannot inject any XSS payload into it. This was the reason when we gave <script>alert()</script> as our input in message field it only showed alert() text in page source because strip_tags() function removed <script> and </script> tags from it.

Second block is for performing input sanitization on Name field. It uses just one function for performing input sanitization. The function is str_replace(). Here this function is replacing all the occurrences of <script> tag with null or blank character. We can use this field to inject our XSS payload. Since this field is replacing all occurrences of <script> tag so we cannot use any XSS payload which contains <script> tag in it.

We can bypass this security by using some other payloads which do not contain <script> tags in it and we can use script tag with different casing enabled. Like we can use <Script> or <scRiPt> or <ScRiPt> in place of <script>. Let us inject it in Name field. So when I tried to enter this payload in Name field it does not accept all the characters of our payload because it has some client side restriction. The restriction is, that a user can enter a maximum of 10 characters in the name field. Since it is a client-side restriction so it can be bypassed very easily.

<scRipt> alert("You have been hacked!"); </scRipt>



Damn Vulnerable Web Application (DVWA) v1.10 *Development*Source :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* — Mozilla Firefox
127.0.0.1/DVWA/vulnerabilities/view_source.php?id=xss_s&security=medium
Vulnerability: Stored Cros
127.0.0.1/DVWA/vulnerabilities/xss_s/
Sign Guestbook
Clear Guestbook
1
2
View Source
View Help

Stored XSS Source

vulnerabilities/xss_s/source/medium.php

```
<?php

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

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($G
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOB
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"]))

    //mysql_close();
}

?>
```

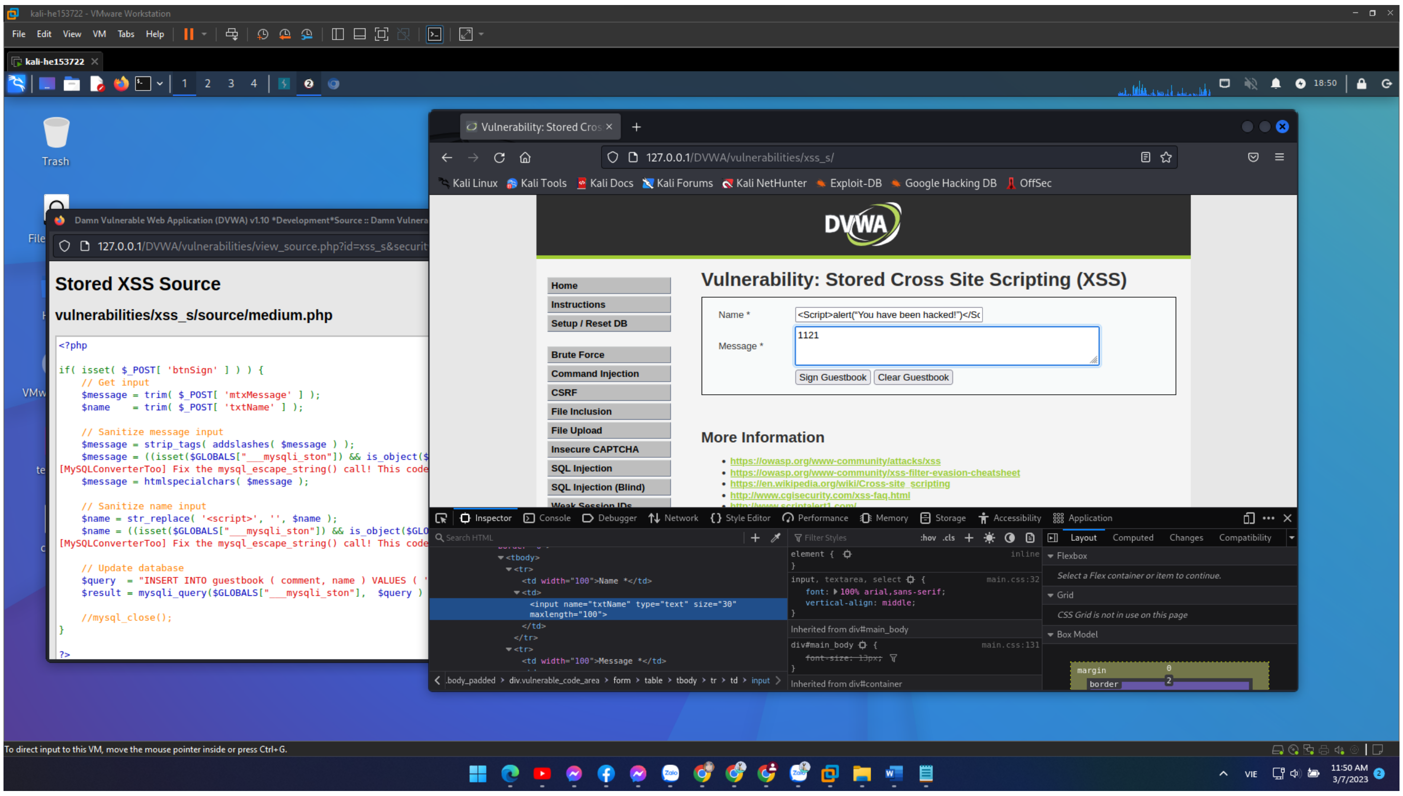
Name: 1
Message:

Name: 2
Message:

More Information

- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Damn Vulnerable Web Application (DVWA) v1.10 *Development*



Stored XSS Source

vulnerabilities/xss_s/source/medium.php

```
<?php

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

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($
[MySQLConverterToo] Fix the mysql_escape_string() call! This code
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLO
[MySQLConverterToo] Fix the mysql_escape_string() call! This code

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query )

    //mysql_close();
}

?>
```

127.0.0.1/DVWA/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

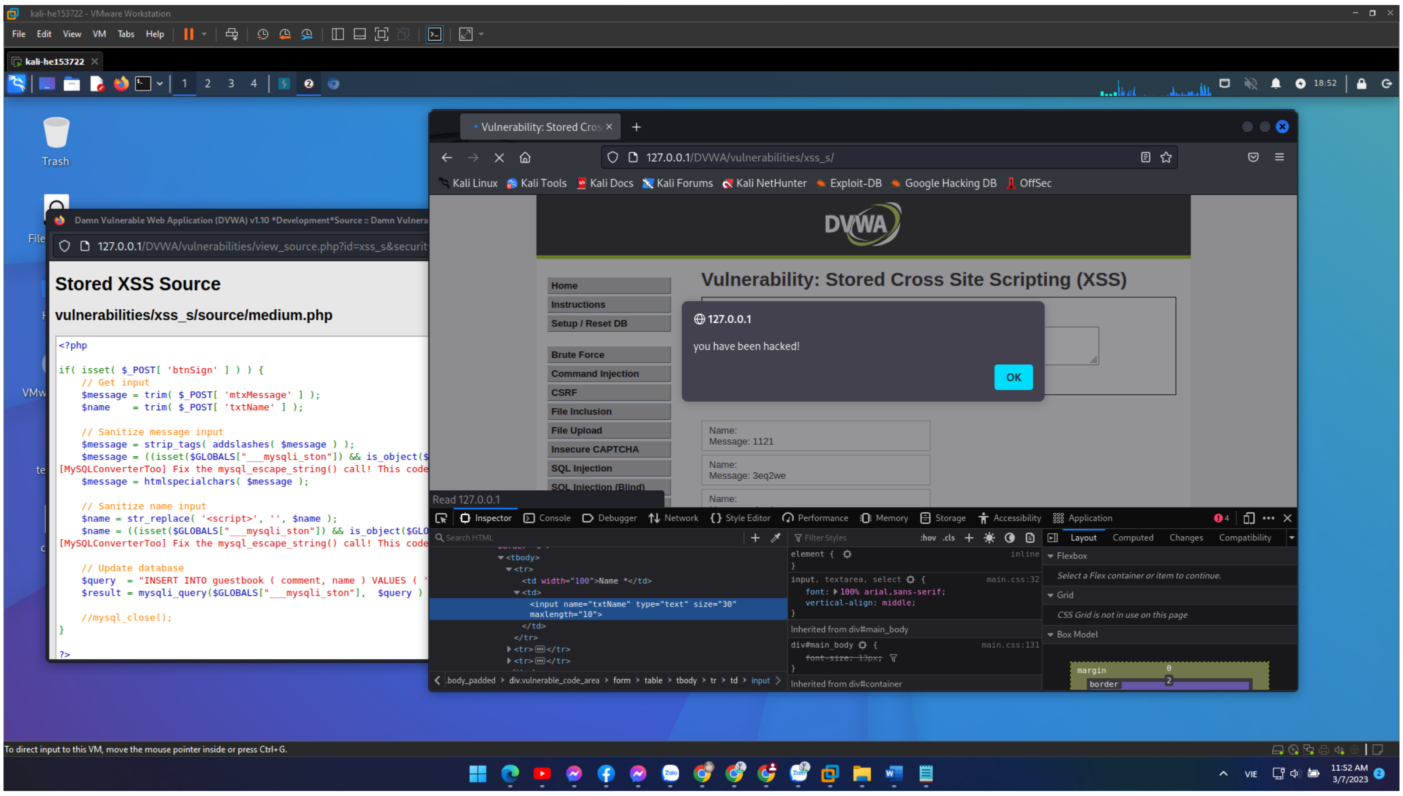
Name * <Script>alert("You have been hacked!")</Sc

Message * 1121

More Information

- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html

```
<td width="100">Name *</td>
<input name="txtName" type="text" size="30" maxlength="100">
```



127.0.0.1

you have been hacked!

Name:
Message: 1121

Name:
Message: 3eq2we

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

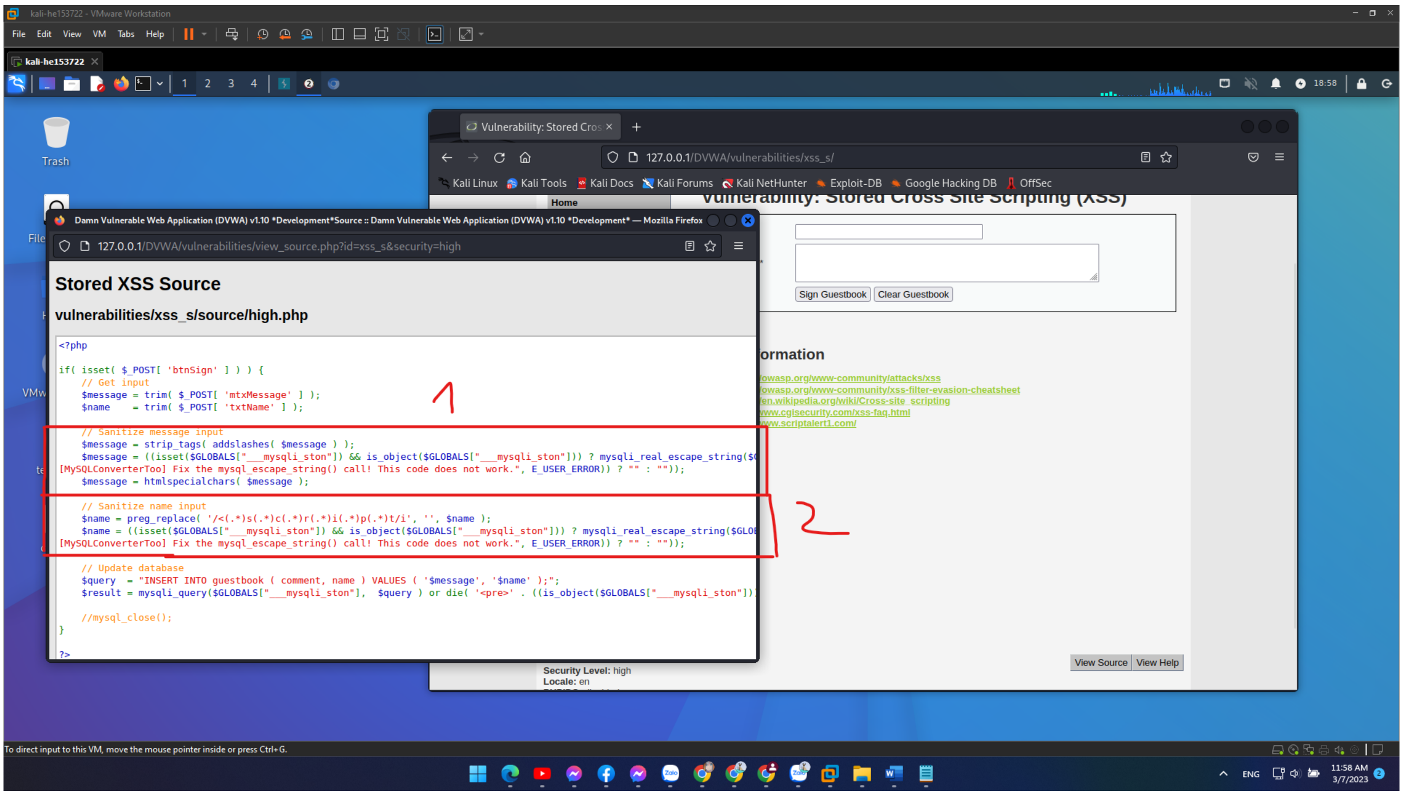
## HIGH

According to source code block 1 is accepting input of Message field and performing sanitization on it. Again this code is using strip_tags() and htmlspecialchars() functions to sanitize user input as we have seen in medium security. So there is no chance to inject XSS payload in this field because they will be filtered by these functions.

Block 2 is for performing input sanitization on Name field. It uses just one function for performing input sanitization. The function is preg_replace(). Here this function replaces every occurrences of <script> tag irrespective of their cases with null or blank. So here every XSS payload which includes <script> tag into it will be useless.

To bypass this, we'll do the same step in the medium section but have to use some other XSS payload that doesn't contain the <script> tag in it.

<img src=X onerror="alert('you have been hacked!')">



Stored XSS Source

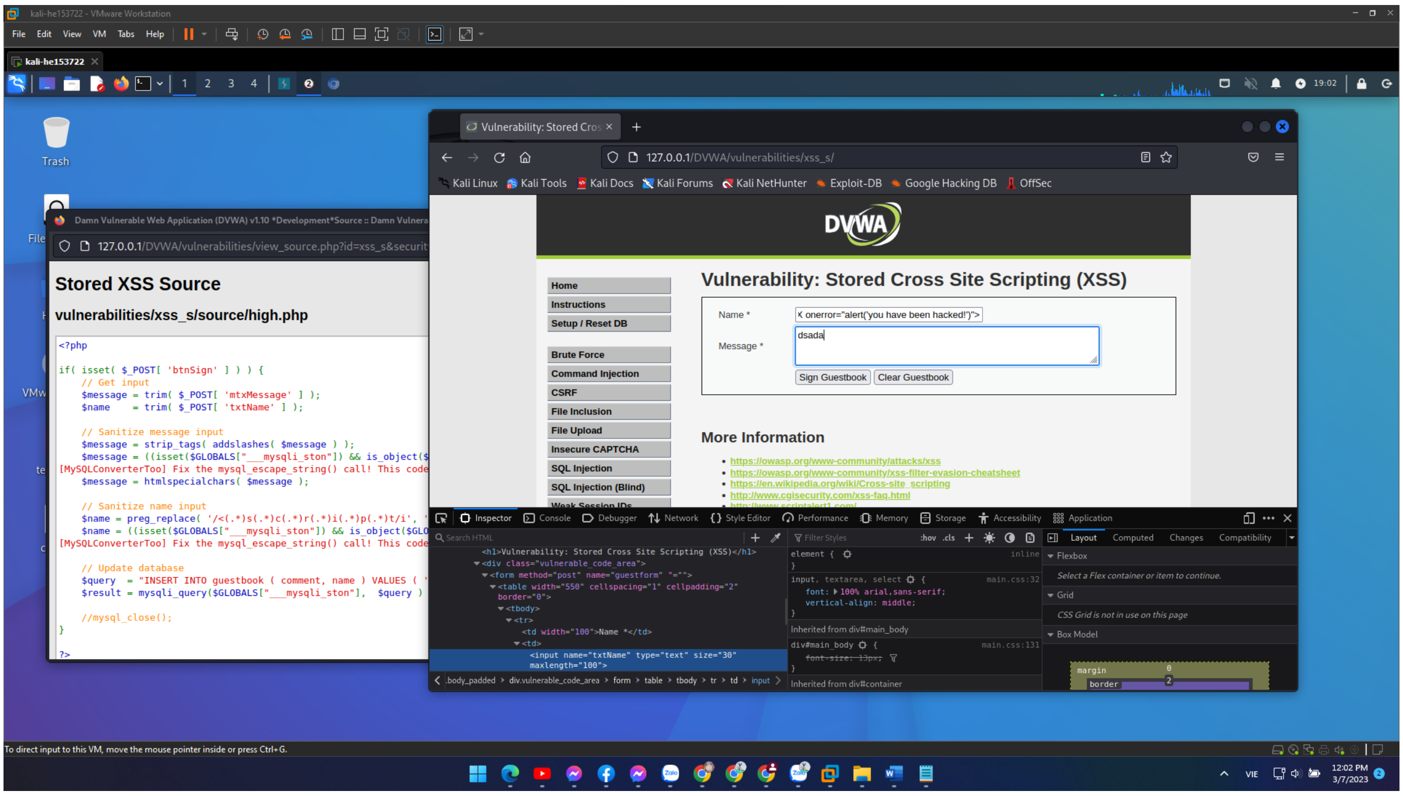
vulnerabilities/xss_s/source/high.php

```php
<?php

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

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($G
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOE
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"]))

    //mysql_close();
}

?>
```

1

2

Security Level: high
Locale: en

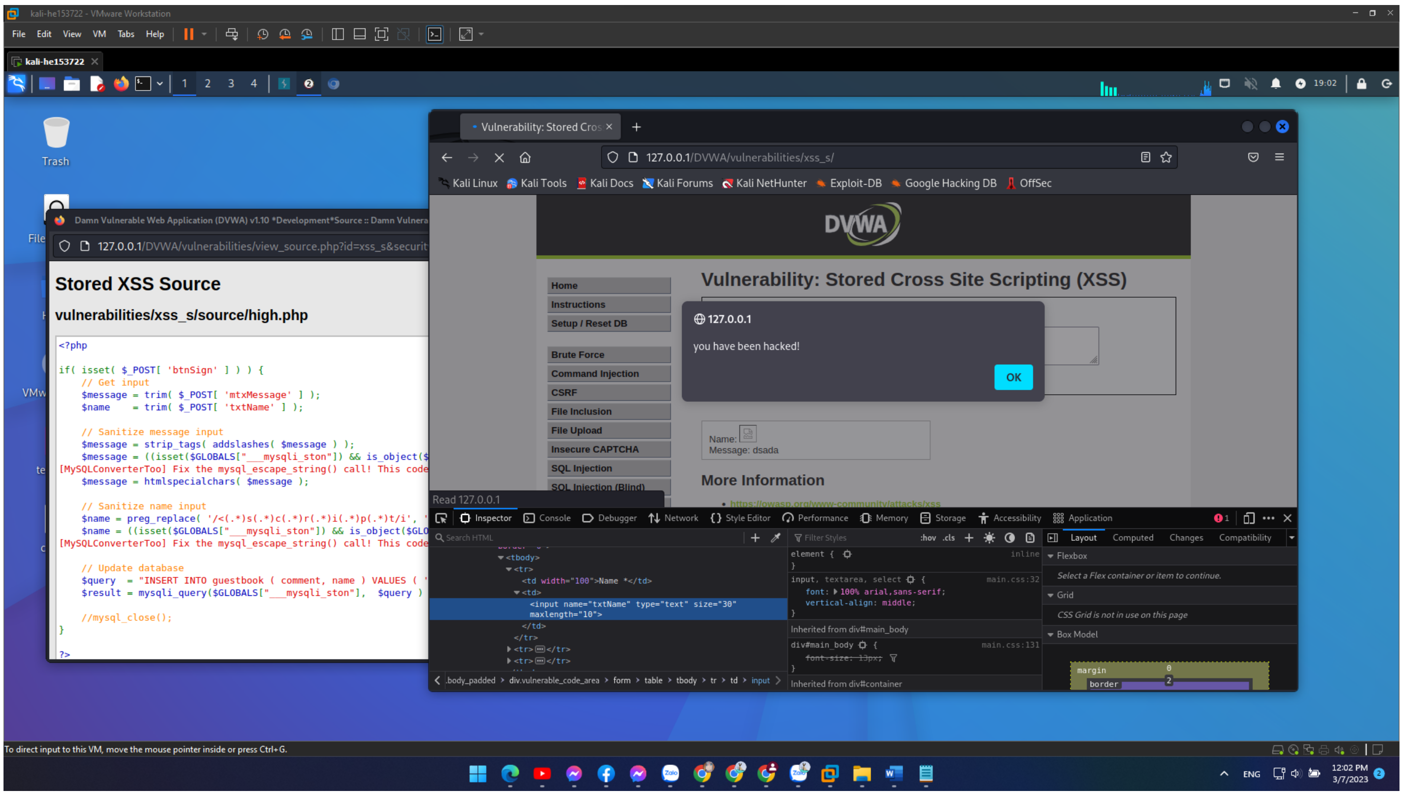


Vulnerability: Stored Cross Site Scripting (XSS)

Name * x onerror="alert('you have been hacked!')">

Message * dsada

```html
<h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
<div class="vulnerable_code_area">
  <form method="post" name="guestform" "="">
    <table width="550" cellspacing="1" cellpadding="2"
    border="0">
      <tbody>
        <tr>
          <td width="100">Name *</td>
          <td>
            <input name="txtName" type="text" size="30"
            maxlength="100">
```



Stored XSS Source

vulnerabilities/xss_s/source/high.php

```
<?php

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

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($
[MySQLConverterToo] Fix the mysql_escape_string() call! This code
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLO
[MySQLConverterToo] Fix the mysql_escape_string() call! This code

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES ( '
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query )

    //mysql_close();
}

?>
```

Vulnerability: Stored Cross Site Scripting (XSS)

127.0.0.1

you have been hacked!

OK

Name:
Message: dsada

More Information

- https://owasp.org/www-community/attacks/xss

```
<tbody>
  <tr>
    <td width="100">Name *</td>
    <td>
      <input name="txtName" type="text" size="30"
      maxlength="10">
    </td>
  </tr>
  <tr>…</tr>
  <tr>…</tr>
```

.body_padded > div.vulnerable_code_area > form > table > tbody > tr > td > input

```
element {
}
input, textarea, select {                  main.css:32
  font: 100% arial,sans-serif;
  vertical-align: middle;
}
Inherited from div#main_body
div#main_body {                            main.css:131
  ~~font-size: 13px;~~
}
Inherited from div#container
```

Flexbox
Select a Flex container or item to continue.
Grid
CSS Grid is not in use on this page
Box Model