

SAD LAB EXPERIMENT - 08

Aim: To study and implement Cross-site Scripting(XSS) vulnerabilities.

To do:

1. What is Cross-site Scripting?
2. Types of XSS attacks.
3. How to prevent XSS attacks?

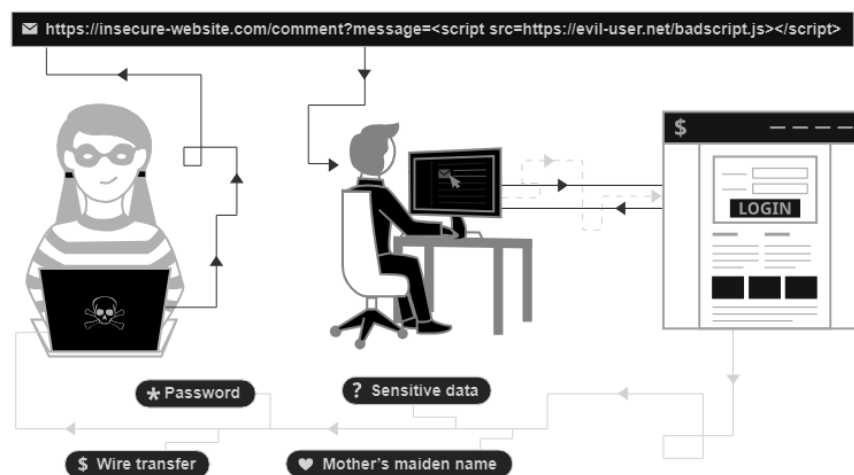
Theory:

Cross-site Scripting

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

Working of XSS:

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.



Types of XSS attacks:

There are three main types of XSS attacks. These are:

- **Reflected XSS**, where the malicious script comes from the current HTTP request.
- **Stored XSS**, where the malicious script comes from the website's database.

- **DOM-based XSS**, where the vulnerability exists in client-side code rather than server-side code.

a. Reflected cross-site scripting

Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Here is a simple example of a reflected XSS vulnerability:

```
https://insecure-website.com/status?message=All+is+well.  
<p>Status: All is well.</p>
```

The application doesn't perform any other processing of the data, so an attacker can easily construct an attack like this:

```
https://insecure-website.com/status?message=<script>/*+Bad+stuff+here...+*/  
</script>  
<p>Status: <script>/* Bad stuff here... */</script></p>
```

If the user visits the URL constructed by the attacker, then the attacker's script executes in the user's browser, in the context of that user's session with the application. At that point, the script can carry out any action, and retrieve any data, to which the user has access.

b. Stored cross-site scripting

Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources; for example, a webmail application displaying messages received over SMTP, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic.

Here is a simple example of a stored XSS vulnerability. A message board application lets users submit messages, which are displayed to other users:

```
<p>Hello, this is my message!</p>
```

The application doesn't perform any other processing of the data, so an attacker can easily send a message that attacks other users:

```
<p><script>/* Bad stuff here... */</script></p>
```

c. DOM-based cross-site scripting

DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

In the following example, an application uses some JavaScript to read the value from an input field and write that value to an element within the HTML:

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

If the attacker can control the value of the input field, they can easily construct a malicious value that causes their own script to execute:

```
You searched for: <img src=1 onerror='/* Bad stuff here... */'>
```

In a typical case, the input field would be populated from part of the HTTP request, such as a URL query string parameter, allowing the attacker to deliver an attack using a malicious URL, in the same manner as reflected XSS.

Finding and testing for XSS vulnerabilities

The vast majority of XSS vulnerabilities can be found quickly and reliably using Burp Suite's web vulnerability scanner.

Manually testing for reflected and stored XSS normally involves submitting some simple unique input (such as a short alphanumeric string) into every entry point in the application, identifying every location where the submitted input is returned in HTTP responses, and testing each location individually to determine whether suitably crafted input can be used to execute arbitrary JavaScript. In this way, you can determine the context in which the XSS occurs and select a suitable payload to exploit it.

Manually testing for DOM-based XSS arising from URL parameters involves a similar process: placing some simple unique input in the parameter, using the browser's developer tools to search the DOM for this input, and testing each location to determine whether it is exploitable. However, other types of DOM XSS are harder to detect. To find DOM-based vulnerabilities in non-URL-based input (such as document.cookie) or non-HTML-based sinks (like setTimeout), there is no substitute for reviewing JavaScript code, which can be extremely time-consuming. Burp Suite's web vulnerability scanner combines static and dynamic analysis of JavaScript to reliably automate the detection of DOM-based vulnerabilities.

How to prevent XSS attacks?

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defence, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Implementation:

a. Reflected cross-site scripting

Step 1: Go to Burp Suite Reflected XSS Labs and Click on Access Lab button: [Link](#)


Lab: Reflected XSS into HTML context with nothing encoded


APPRENTICE

This lab contains a simple reflected cross-site scripting vulnerability in the search functionality.

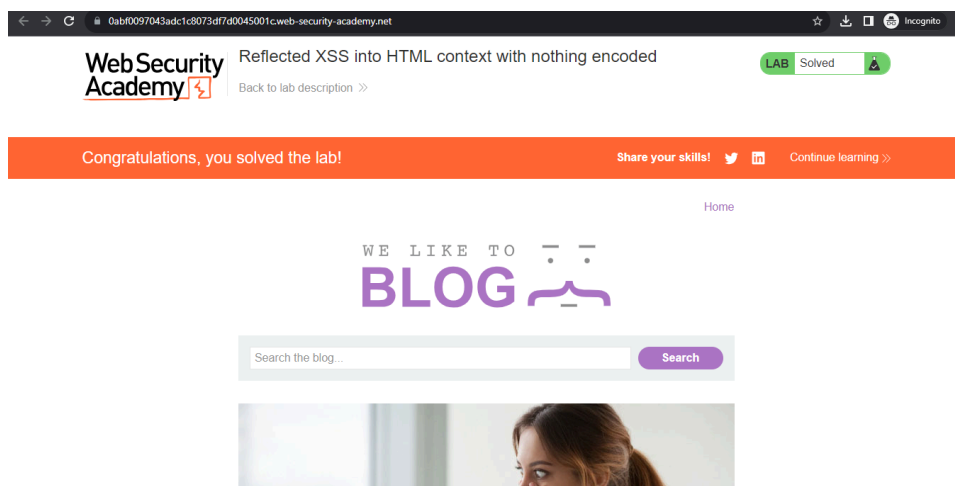
To solve the lab, perform a cross-site scripting attack that calls the `alert` function.

 ACCESS THE LAB

 Solution

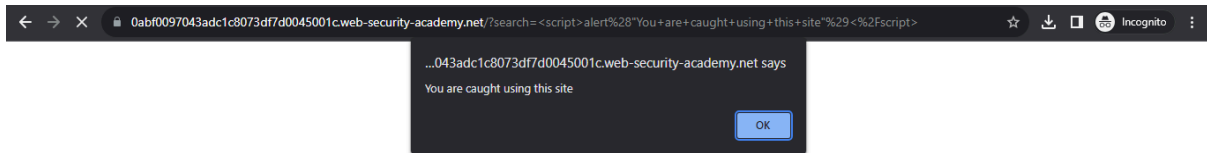
 Community solutions

Step 2: Search for `<script>alert("You are caught using this site")</script>` in the search column.

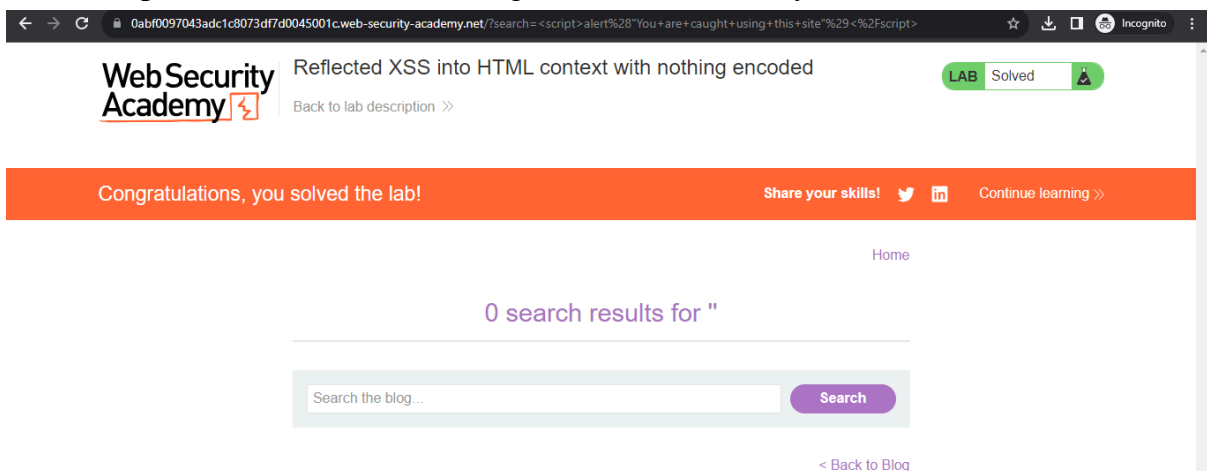


Search

Step 3: The application echoes the supplied search term in the response to this URL:



Step 4: Reflected XSS attack is performed successfully:



b. Stored cross-site scripting

Step 1: Go to Burp Suite Stored XSS Labs and Click on Access Lab button: [Link](#)

Lab: Stored XSS into HTML context with nothing encoded



This lab contains a stored cross-site scripting vulnerability in the comment functionality.

To solve this lab, submit a comment that calls the `alert` function when the blog post is viewed.



Step 2: Once you click a new tab will get opened. Scroll down and click on View post on any one of the posts.

WebSecurity Academy


Stored XSS into HTML context with nothing encoded

LAB Not solved

[Back to lab description >>](#)

Home

WE LIKE TO BLOG



Faking It! - InstaCam


People are going to extreme lengths to pull the wool over their friends' eyes on Social Media. If you've ever clicked your way through family photos and the perfect summer and winter getaway pics of your friends on Instagram then...

[View post](#)

Step 3: The post will open and now scroll down to the Leave a comment section

0a5d003304eae5ad866a534000fb0037.web-security-academy.net/post?postId=6

☆ ⬇️ 🗨️ Incognito



Faking It! - InstaCam

El Bow | 14 August 2023

People are going to extreme lengths to pull the wool over their friends' eyes on Social Media. If you've ever clicked your way through family photos and the perfect summer and winter getaway pics of your friends on Instagram then you'll know why.

There you are, sitting all alone apart from your six cats, parent, and a spider called Shan. Your

Leave a comment

Comment:

Name:

Email:

Website:

Post Comment

[< Back to Blog](#)

Step 4: Now fill the comment, name, email and website of your name:

Comment: `<script>alert("You are caught!!!")</script>`

Name: Hacker007

Email: hacker007@gmail.com

Website: https://hacker007.com

Leave a comment

Comment:

```
<script>alert("You are caught!!!")</script>
```

Name:

Hacker007

Email:

hacker007@gmail.com

Website:

https://hacker007.com

Post Comment

[< Back to Blog](#)

Step 5: Once you post the comment, it will show a pop up that the comment is done successfully. Now when you click on the back to block button, it will give an alert window that you are caught.

WebSecurity Academy

Stored XSS into HTML context with nothing encoded

LAB Not solved

[Back to lab description >>](#)

Home

Thank you for your comment!

Your comment has been submitted.

[< Back to blog](#)

...04eae5ad866a534000fb0037.web-security-academy.net says

You are caught!!!

OK

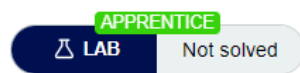
Step 6: It is successfully executed

 **Hacker007** | 11 September 2023

c. DOM-based cross-site scripting

Step 1: Go to Burp Suite DOM-based XSS Labs and Click on Access Lab button:
[Link](#)

Lab: DOM XSS in `document.write` sink using source `location.search`

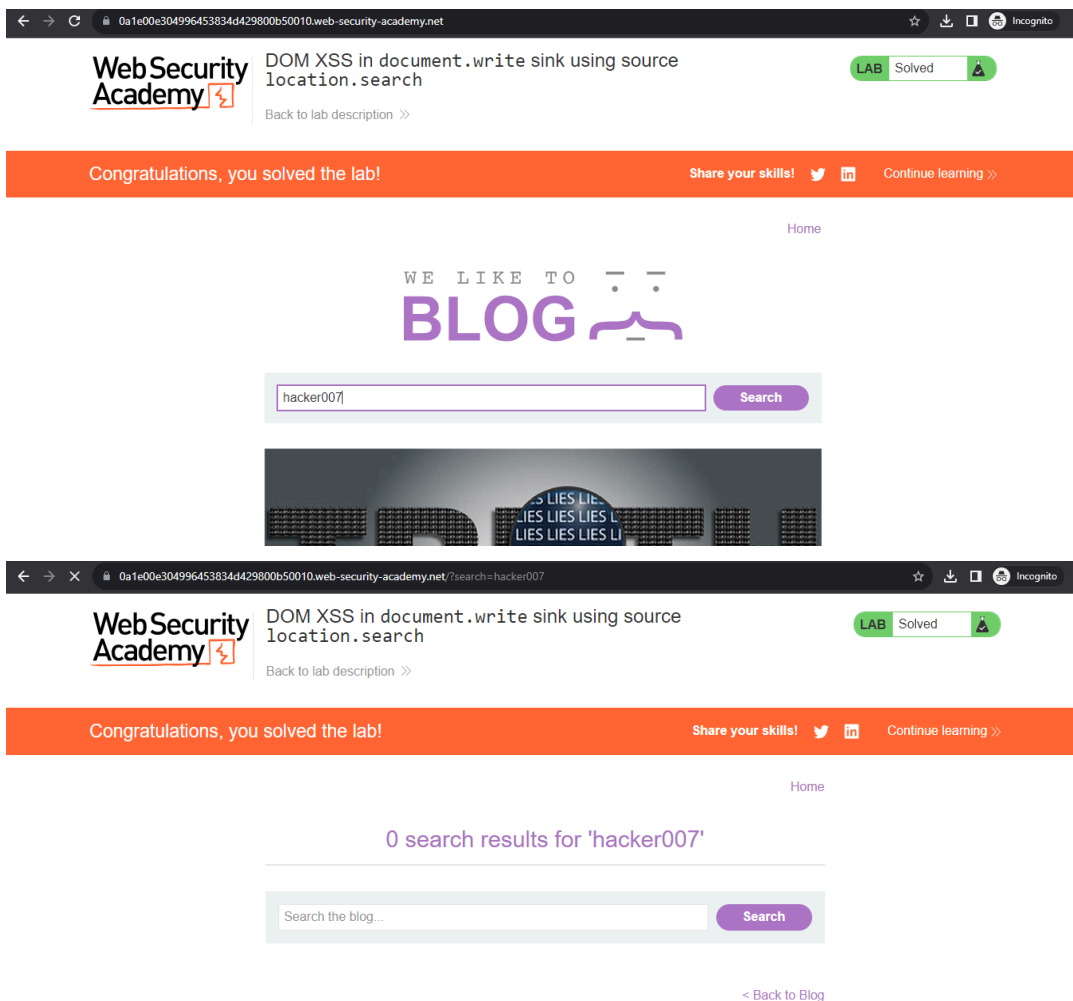


This lab contains a DOM-based cross-site scripting vulnerability in the search query tracking functionality. It uses the JavaScript `document.write` function, which writes data out to the page. The `document.write` function is called with data from `location.search`, which you can control using the website URL.

To solve this lab, perform a cross-site scripting attack that calls the `alert` function.



Step 2: Once you click a new tab will get opened. Now search for any blog.



Step 3: Now search for "><svg onload=alert(1)>

The screenshot shows the Web Security Academy interface. At the top, a navigation bar includes the logo, a title 'DOM XSS in document.write sink using source location.search', a 'LAB Solved' badge, and a 'Back to lab description' link. Below this is an orange banner with the text 'Congratulations, you solved the lab!' and links to 'Share your skills!', social media icons, and 'Continue learning >>'. The main content area has a 'Home' link and a search bar with the text '0 search results for 'hacker007''. Below the search bar is a search input field containing '><svg onload=alert(1)>' and a 'Search' button. At the bottom, there is a '< Back to Blog' link.

Step 4: The attack is done successfully

The screenshot shows the Web Security Academy interface after a successful attack. A dark alert box is displayed in the center, containing the text '...04996453834d429800b50010.web-security-academy.net says' and an 'OK' button. The background shows the same navigation bar and banner as in Step 3, but the search results are not visible.

Conclusion:

We successfully studied and implemented Cross-site Scripting(XSS) attacks