

Forum:[Hack Forums](#)

Thread: [XSS for Beginners A Practical Lab Guide](#)

ICTKUN - September 23, 2026, 2:01 pm

Cross-Site Scripting (XSS) happens when an application takes untrusted input and places it into a web page in a way that causes the browser to interpret that input as code rather than ordinary data. OWASP describes three major categories: reflected XSS, stored XSS, and DOM-based XSS.

For learning, we'll use a deliberately vulnerable application such as OWASP Juice Shop, which is specifically designed for security training and CTF-style practice.

I'll keep the practical exploitation inside a local lab rather than walking through attacking arbitrary websites.

1. First understand the browser

Imagine a website has:

Code `<h1>Welcome, USER_INPUT</h1>`
The developer expects someone to enter:

Code `ASS`
The browser receives:

Code `<h1>Welcome, ASS</h1>`
Nothing unusual.

But suppose the input is:

? loading encrypted image...
and the application inserts it directly into HTML.

The browser may interpret:

? loading encrypted image...
The important concept is:

Code `INPUT`
?
WEB APPLICATION
?
HTML RESPONSE / DOM
?
BROWSER PARSES IT
?
JAVASCRIPT EXECUTES

That's the basic XSS pipeline.

2. The three main types

Reflected XSS

The malicious input comes through a request and is immediately reflected into the response.

Conceptually:

Code Browser

?

?q=<payload>

?

Server

?

HTML response containing payload

?

Browser executes it

Example vulnerable server logic:

? loading encrypted image...

A normal request:

Code /search?q=shoes

produces:

Code <h1>Search results for shoes</h1>

But an XSS test value such as:

? loading encrypted image...

would be inserted directly.

The fundamental problem isn't the word script.

It's this:

Code \${query}

is being placed into an HTML execution context without appropriate output encoding.

3. Stored XSS

Stored XSS is different.

The payload gets saved somewhere first.

For example:

Code Browser

?

Comment form

?

Server

?

Database

?

Comment displayed later

?

Browser

?

JavaScript executes

Imagine a vulnerable comment system:

```
Code app.post("/comment", (req, res) => {  
  database.save(req.body.comment);  
});
```

Later

```
Code app.get("/comments", (req, res) => {  
  const comments = database.getAll();  
  
  res.send(comments.map(c =>  
    `

${c}</div>`  
  ).join(""));  
});


```

Someone submits:

? loading encrypted image...

The database stores it.

Every time the comments page renders that value, the browser may execute it.

That's why stored XSS can be particularly significant: the malicious input persists rather than existing only inside one request.

4. DOM-based XSS

This one happens primarily inside client-side JavaScript.

For example:

? loading encrypted image...

The application reads data from the URL:

Code location.hash
and sends it into:

Code innerHTML
That's a dangerous source -> sink relationship.

OWASP specifically identifies APIs such as innerHTML, outerHTML, and document.write() as dangerous when supplied with untrusted data.

A safer version is:

```
Code const input = location.hash.substring(1);
```

```
document.getElementById("output").textContent = input;
```

5. Your first local XSS laboratory

Create

Code xss-lab/
+-- index.html
Put this inside :

[Pastebin link code](#)

Open it in your browser.

Enter:

Code hello
You'll see:

Code hello
Now, inside this local laboratory, enter:

? loading encrypted image...
If the browser executes the JavaScript, you've demonstrated XSS.

6. Why did it happen?

Trace the data:

Code const input = document.getElementById("input").value;
?

Code input
?

Code document.getElementById("output").innerHTML = input;
?

Code UNTRUSTED INPUT
?
innerHTML
?

HTML PARSER
?

JAVASCRIPT

This is the fundamental pattern you want to learn to recognize.

7. Source and sink

Security researchers often think about XSS as:

SOURCE -> TRANSFORMATION -> SINK

Sources

Places where attacker-controlled data can originate:

Code location.search
location.hash
location.href
document.referrer
localStorage
sessionStorage
URL parameters
form fields
POST data
database values
Dangerous sinks

Places where the data can become executable:

Code innerHTML
outerHTML
document.write()
document.writeln()
eval()
setTimeout(string)
setInterval(string)
The exact danger depends on the context.

8. Finding XSS in a lab

Don't start by throwing random payloads everywhere.

Start by identifying where your input travels.

Suppose your lab has:

Code /search?q=
Enter :

Code TEST123
Then inspect the resulting page.

You're looking for:

Code TEST123
If it appears somewhere, determine how it appears.

For example:

? loading encrypted image...
means your value entered HTML text.

But:
? loading encrypted image...
means your value entered an HTML attribute.

And:

Code `const name = "TEST123";`
means your value entered JavaScript.

These are different contexts.

That's one of the most important lessons in XSS testing.

9. Context matters

Imagine:

? loading encrypted image...

versus:

? loading encrypted image...

versus:

Code `const x = "USER_INPUT";`

They look similar conceptually, but the browser parses them differently.

Therefore:

Code HTML context

!=

Attribute context

!=

JavaScript context

!=

URL context

!=

CSS context

OWASP emphasizes context-sensitive output encoding because different HTML contexts require different handling.

CyberXero - September 23, 2026, 2:36 pm

Quote: [ICTKUN](#) For learning, we'll use a deliberately vulnerable application such as OWASP Juice Shop, which is specifically designed for security training and CTF-style practice.

Honestly this guide looks detailed but its quite all over the place, if you mention we will use a deliberately vulnerable application such as OWASP juice shop, then base the guide on that because your thread title states "XSS for beginners A Practical Lab Guide..

1) its not beginner friendly, they dont even know where to get started

2) you havn't mentioned how to setup the Practical lab

at this point i think your post whoring for the 10 hacking guide achievement

LQ shit bruh