# XSS attacks

# Definition

XSS (Cross-Site Scripting) attacks are security vulnerabilities in web applications where an attacker injects malicious scripts into trusted websites, allowing them to execute arbitrary code in the victim's browser. This can lead to unauthorized access, data theft, cookie hijacking, and website defacement.

*Classic example : An attacker submits a comment on a forum with a malicious script embedded in it with

```
<script>
  //get user's cookie
</script>
```

When other users view the comment, the script gets executed in their browsers, giving the attacker access to their sensitive information or allowing them to perform actions on their behalf.*

# Security risks

Security risks associated with XSS attacks include:

1. **Data theft**: Attackers can steal sensitive information, such as login credentials, personal data, or financial details, from users who unknowingly execute the malicious scripts.
2. **Cookie hijacking**: By injecting scripts, attackers can access or manipulate user cookies, leading to session hijacking or impersonation.
3. **Session riding**: Attackers can exploit XSS vulnerabilities to piggyback on authenticated sessions and perform actions on behalf of users, leading to unauthorized access and privilege escalation.
4. **Malware distribution**: Attackers can use XSS to deliver malware or malicious payloads to unsuspecting users, leading to system compromise or further attacks.
5. **Phishing attacks**: XSS vulnerabilities can be leveraged to create convincing phishing pages or pop-ups, tricking users into disclosing sensitive information or installing malicious software.

# How to prevent it

There is 3 solutions that can be used altogether to prevent XSS attacks :

- **Input validation**: Validate and sanitize all user input on the server-side to ensure it does not contain malicious code or script tags.

> **All user's input, event file's name must be sanitized.**

- **Output encoding**: Encode user-generated content before displaying it on web pages to prevent the browser from interpreting it as executable code. Apply appropriate encoding based on the context in which the output is being used (e.g., HTML, JavaScript, CSS) to prevent script injection.

> **Always use your framework var injection with** `{}` which encode for you the variable content.
> Obviously, never inject your variables in `dangerouslySetInnerHTML` unless you want to inject js, that cannot be controlled by the user. But it is really **not recommended.**

- **Content Security Policy (CSP)**: Implement a robust CSP that restricts the types of content allowed to be loaded on a web page, limiting the potential sources of XSS attacks. Here, we inform the browser that the only source able to load js will be your website's subdmain.
  Adapt this header with your specifics needs :

```
Content-Security-Policy: script-src 'self' *.example.com;
```

---

Revision #2
Created 2024-04-12 02:42:56 UTC by Seaweedbrain
Updated 2024-04-12 02:58:32 UTC by Seaweedbrain