Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is one of the most popular and vulnerable attacks known by every advanced tester. It is considered one of the riskiest attacks on web applications and can bring harmful consequences too.
XSS is often compared with similar client-side attacks, as client-side languages are primarily used during this attack. However, an XSS attack is considered riskier because of its ability to damage even less vulnerable technologies.

A cross-site Scripting attack is a malicious code injection that will be executed in the victim's browser. The malicious script can be saved on the web server and executed whenever the user calls the appropriate functionality. It can also be performed with the other methods – without any saved script in the web server.

The primary purpose of this attack is to steal the other user's identity data – cookies, session tokens, and other information. In most cases, this attack steals the other person's cookies. As we know, cookies help us to log in automatically. Therefore with stolen cookies, we can log in with the other identities. Moreover, this is one of the reasons this attack is considered one of the riskiest attacks.

An XSS attack is being performed on the client side. It can be performed with different client-side programming languages. However, most often, this attack is performed with Javascript and HTML.

How is XSS Being Performed?
Cross-Site Scripting attack means sending and injecting malicious code or script. Malicious code is usually written with client-side programming languages such as Javascript, HTML, VBScript, Flash, etc. However, Javascript and HTML are primarily used to perform this attack.
This attack can be performed in different ways. For example, depending upon the type of XSS attack, the malicious script may be reflected on the victim's browser or stored in the database and executed every time the user calls the appropriate function.

The main reason for this attack is improper user input validation, where malicious input can get into the output. For example, a malicious user can enter a script, which will be injected into the website's code. Then the browser cannot know if the executed code is malicious.

Therefore malicious script is being executed on the victim's browser, or any faked form is displayed for the users. There are several forms in which XSS attacks can occur.

The main forms of Cross Site Scripting are as follows:

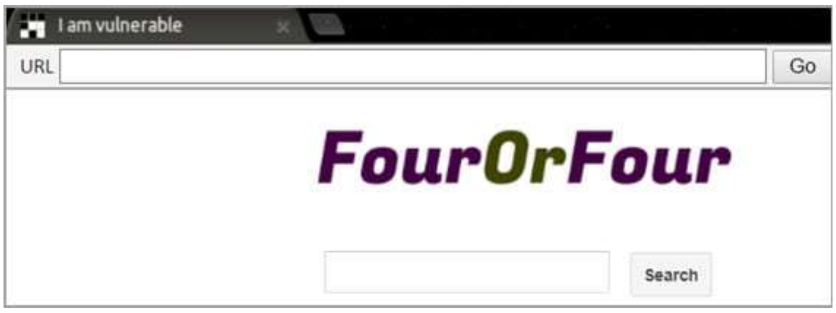
- Cross-Site Scripting can occur on the malicious script executed on the client side.
- Fake page or form displayed to the user (where the victim types credentials or clicks a malicious link).
- On websites with displayed advertisements
- Malicious emails sent to the victim

This attack occurs when the malicious user finds the vulnerable parts of the website and sends it as appropriate malicious input. Then, a malicious script is injected into the code and sent to the final user as output.

XSS Example

1. **Example 1**

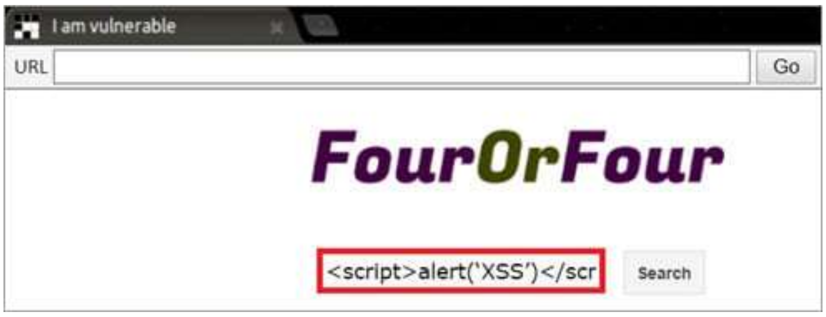
Let us analyze a simple Example: Consider our website with a search field.



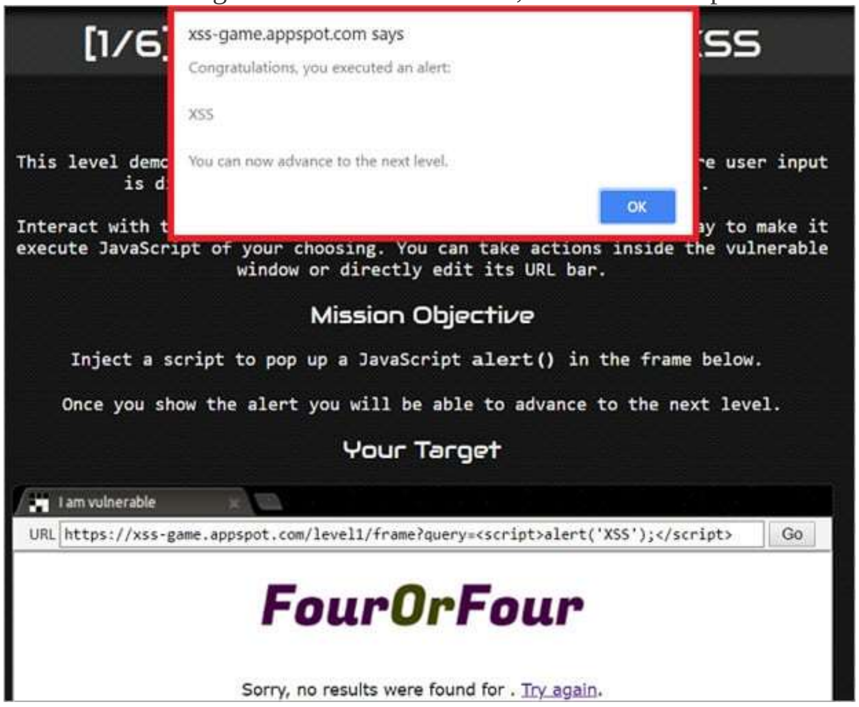
If the search field is vulnerable, when the user enters any script, then it will be executed.

Consider a user enters a very simple script as shown below:

<script>alert('XSS')</script>



Then after clicking on the **"Search"** button, the entered script will be executed.



xss-game.appspot.com says

Congratulations, you executed an alert:

XSS

You can now advance to the next level.

OK

[1/6 SS

This level demc e user input is d .

Interact with t ay to make it execute JavaScript of your choosing. You can take actions inside the vulnerable window or directly edit its URL bar.

Mission Objective

Inject a script to pop up a JavaScript alert() in the frame below.

Once you show the alert you will be able to advance to the next level.

Your Target

I am vulnerable

URL https://xss-game.appspot.com/level1/frame?query=<script>alert('XSS');</script> Go

FourOrFour

Sorry, no results were found for . Try again.

As the example shows, the script typed into the search field gets executed. This shows the vulnerability of the XSS attack. However, a more harmful script may be typed as well.
Many testers mix up Cross Site Scripting attacks with Javascript Injection, which is also being performed on the client side. In both, the attack's malicious script is being injected. However, in the XSS attack case, <script> tags are unnecessary to execute the script.

For Example:
<body onload=alert('something')>;

Also, it can be a script executed on the other event.

For Example: On a mouse hover.
<b onmouseover=alert('XSS testing!')></b>

2. **Example 2**

Let us analyze another Example: Consider we have a page displaying the latest book review on the website.

The code of this page will look as shown below:

```
print "<html>"
print "<h1>Latest book review</h1>"
print database.latestReview
print "</html>"
```

Therefore, if a malicious user types something harmful in the review field, it will be loaded on this page.

For Example: Consider that in the review field, if a hacker types the below code.

```
<script>destroyWebsite();</script>
```

Then on the page load function destroyWebsite(), it would be called and perform its harmful actions.

As most of us know, this attack is mainly used to gather the other person's cookies, which can be used to log in with the other identities. So, let us analyze another example of a possible XSS script with possible cookies theft.

For example, the hacker injects the appropriate code through the vulnerable website's field.

```
<script type=" text/javascript">

var test='../example.php?cookie_data='+escape(document.cookie);

</script>
```

As the indicated example shows, cookies are escaped and sent to the example.php script's variable 'cookie_data'. Therefore, if the malicious user injects this script into the website's code, it will be executed in the user's browser, and cookies will be sent to the malicious user.

Types of Cross-Site Scripting Attacks
The prime purpose of performing an XSS attack is to steal another person's identity. As mentioned, it may be cookies, session tokens, etc. XSS also may display fake pages or forms for the victim. However, this attack can be performed in several ways.

This attack is divided into three main categories, as shown below:

1) Reflected XSS – This attack occurs when a malicious script is not saved on the web server but reflected in the website's results.

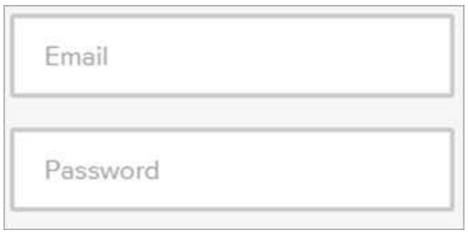
2) Stored XSS – This attack occurs when a malicious script is permanently saved on the webserver.

3) DOM – This occurs when the DOM environment changes, but the code remains the same.

1) Reflected XSS
This occurs when the malicious results are returned after entering the malicious code. Reflected XSS code is not being saved permanently. In this case, the malicious code is reflected in any website result. The attack code can be included in the faked URL or HTTP parameters.

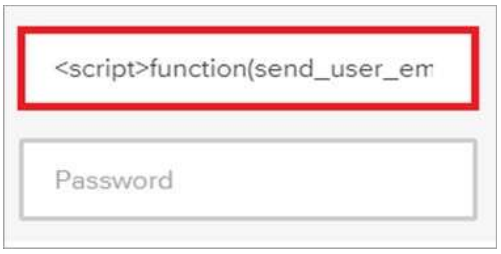
It can affect the victim differently – displaying fake pages or sending a malicious email.

Let us analyze an Example: **We have a login page where the user must type his username and password.**



When wrong credentials are typed into some websites, an error message like "Sorry, your username or your credentials are wrong" will be displayed.

In this **example,** the username is a parameter that is typed by the user in the login form. Including the username parameter in the output is a mistake. This way, an attacker can type the malicious script instead of the correct username or email address.



For example, a script may be sent to the user's malicious email, where the victim may click the faked link.

2) Stored XSS

This attack can be considered riskier, and it provides more damage.

In this attack, the malicious code or script is saved on the webserver (for example, in the database) and executed whenever the users call the appropriate functionality. This way, stored XSS attacks can affect many users. Also, the script stored on the web server will affect the website longer.

In order to perform stored XSS attacks, the malicious script should be sent through the vulnerable input form (For example, the comment field or review field). This way, the appropriate script will be saved in the database and executed on the page load or appropriate function calling.

Consider that we have a page where the latest user opinion is loaded. Therefore, the opinion or comment field would be typed with the script, as shown below.
<script>alert(document.cookie)</script>

It will be saved in the database and executed on the page load, as the latest user opinion will be displayed. If a website is vulnerable to XSS, the page load popup window with cookies will be displayed. This script is quite simple and less harmful. However, a more harmful code may be entered instead of this script.

For example, cookies may be sent to the malicious user, or a fake page may be displayed in the victim's browser.

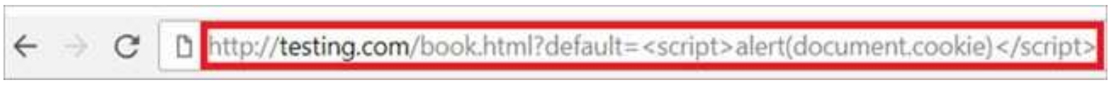
3) DOM XSS

This attack occurs when the DOM environment is being changed, but the client-side code does not change. However, the client-side code executes differently when the DOM environment is being modified in the victim's browser.
In order to get a better understanding of how the XSS DOM attack is being performed, let us analyze the following example.

Consider there is a webpage with the URL *http://testing.com/book.html?default=1*. As we know, "default" is a parameter, and "1" is its value. Therefore, we would send a script as the parameter to perform an XSS DOM attack.
For Example:
http://testing.com/book.html?default=<script>alert(document.cookie)</script>

In this example, the request is sent for the page *book.html?default=<script>alert(document.cookie)</script>* to *testing.com*.

Therefore, the browser creates a DOM object for that page, where the document location object will contain the appropriate string.
http://testing.com/book.html?default=<script>alert(document.cookie)</script>


http://testing.com/book.html?default=<script>alert(document.cookie)</script>

This way, the DOM environment is being affected. However, something more harmful may also be entered instead of this simple script.

How to Test Against XSS?

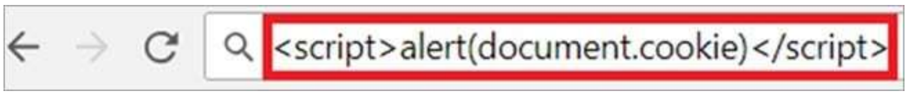
Firstly, in order to test against XSS attacks, black box testing can be performed.
It means that it can be tested without a code review. However, a tester should consider which website's parts are vulnerable to the possible XSS attack during testing.

It is better to list them in any testing document; this way, we will be sure that nothing will be missed. Then, the tester should plan for what code or script input fields must be checked. It is important to remember what results mean, that the application is vulnerable, and that it analyzes the results thoroughly.

While testing for a possible attack, it is important to check how it responds to the typed scripts, whether they are executed, etc.

For example, a tester may try to type in the browser script like:

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

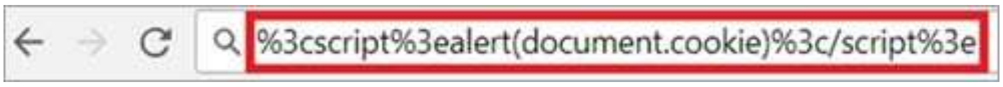


```
<script>alert(document.cookie)</script>
```

If this script is being executed, then there is a considerable possibility that XSS is possible.

Also, while testing manually for possible Cross Site Scripting attacks, it is essential to remember that encoded brackets should also be tried.

For Example:
%3cscript%3ealert(document.cookie)%3c/script%3e



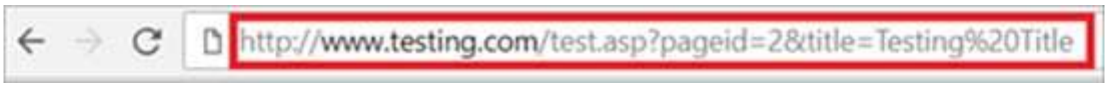
```
%3cscript%3ealert(document.cookie)%3c/script%3e
```

Some try to protect the websites and systems from attacks by changing the brackets into double.

For example, if the input field is typed with bracket "<", then it would be changed to double "<<". Therefore, it is essential to remember that testing with encoded brackets should also be executed.

It would be best not to forget to test the website's URL. For example, we have a request:

http://www.testing.com/test.asp?pageid=2&title=Testing%20Title



```
http://www.testing.com/test.asp?pageid=2&title=Testing%20Title
```

If this attack is possible, the HTML code will include <h1>Testing Title</h1>. Additionally, if this vulnerability is present in the web application, an indicated text will be inserted in <h1></h1> tags.
Passing some code through an HTTP request is also a method to check if this attack is possible.

Generally, input validation should be checked while testing for possible XSS attacks, and the tester should be conscious while checking the website's output. Also, if a code review is being performed, finding how input can get into the output is essential.

XSS Testing Tools

As a Cross-Site Scripting attack is one of the most popular risky attacks, many tools exist to test it automatically. We can find various scanners to check for possible XSS attack vulnerabilities – like, Nessus and Nikto. Both of which are considered quite reliable.

SOAP UI can be considered a potent tool for checking against possible XSS attacks. This is because it contains ready templates for checking against this attack. In addition, it simplifies the testing process.
However, in order to test for this vulnerability with the SOAP UI tool, API-level testing should already be automated with that tool. Another solution to test against XSS can be browser plugins. However, plugins are considered quite a weak tool to check against this attack.

Even while testing automatically, the tester should have good knowledge of this attack type and be able to analyze the results appropriately.
Good knowledge is also helpful while selecting the testing tool. Also, it is essential to know that while scanning for security vulnerabilities with an automatic tool, testing manually is also good practice; this way, the tester can see and analyze the results.

Comparison with Other Attacks

XSS is considered one of the riskiest attacks, as its main purpose is to steal the website's or system's user identities. Also, XSS attacks can be performed with different client-side languages like Javascript, HTML, VBScript, Flash, etc. Moreover, this makes it more harmful and widespread than the other possible attacks.
Testing for XSS attacks is similar to testing for other possible client-side attacks. However, it is essential to remember what additional cases should be checked while testing for XSS.
Another thing that makes this attack riskier is the possibility of being stored in the web service – this way, it can affect many users for a longer time. XSS can sometimes be performed on even less vulnerable systems, and its vulnerabilities are sometimes challenging.
Also, compared with the other attacks, XSS has many ways to be performed and affects the website.

Ways to Prevent XSS

Though this type of attack is considered one of the most dangerous and risky, a prevention plan should be prepared. Because of this attack's popularity, there are many ways to prevent it.
Commonly used main prevention methods include:

- Data validation
- Filtering
- Escaping

The first step in the prevention of this attack is **Input validation**. Everything the user enters should be precisely validated because their input may reach the output. Data validation can be named as the basis for ensuring the system's security.

Another suitable prevention method is **user input filtering**. The idea of filtering is to search for risky keywords in the user's input and remove them or replace them with empty strings.
Those keywords may be:
<script></script> tags
Javascript commands
HTML markup

Input filtering is relatively easy to practice. It can be performed in different ways too.
Like:
By developers who have written server-side code.

An appropriate programming language library is being used.
In this case, some developers write their code to search for appropriate keywords and remove them. However, the easier way would be to select an appropriate programming language library to filter the user's input.

Another possible prevention method is **characters escaping**. In this practice, appropriate characters are changed by special codes. For example, < escaped character may look like &#60. Knowing that we can find appropriate libraries to escape the characters is essential.
Meanwhile, good testing should not be forgotten as well. Instead, it should be invested in good software testers' knowledge and reliable software testing tools. This way, good software quality will be better assured.

Prevention According to Technologies

As already discussed, filtering and characters escaping are the main prevention methods. However, it can be performed differently in different programming languages. Some programming languages have appropriate filtering libraries, and some do not.

It should be mentioned that filtering can be performed quite easily in Java and PHP programming languages, as they have appropriate libraries.

Java technology is quite widely used. Therefore there are many solutions to it. If you use Spring technology and want to escape HTML for the whole application, you must write the appropriate code in the project's web.xml file.

```
<context-param>
<param-name>defaultHtmlEscape</param-name>
<param-value>true</param-value>
</context-param>
```

This code will switch HTML escaping for the entire application.

If you would like to switch HTML escaping for the appropriate page's forms, then the code should be written as follows:

```
<spring:htmlEscape defaultHtmlEscape="true" />
```

There are many ready XSS filters in the form of a .jar file. The .jar file must be added to your project, and only its libraries can be used. One such XSS filter is xssflt.jar, which is a servlet filter. This .jar file can be easily downloaded online and added to your project.

This filter checks every request sent to the application and cleans it from a potential injection.

When an external.jar file is added to the project, it also has to be described in the web.xml file:

```
<filter>
<filter-name>XSSFilter</filter-name>
<filter-class>com.cj.xss.XSSFilter</filter-class>
</filter>
```

Another possible solution is the ESAPI library. ESAPI library is compatible with many programming languages. For example, ESAPI libraries for Java and PHP programming languages can be found. In addition, it is an open-source and free library that helps control the application's security.