**XSS into JavaScript:**

When we encounter a sink in JavaScript of JQuery

Text

Description automatically generatedSomething like this if we have control over the author or poem variable could lead to XSS. Start by looking for that reflection!

So here we could enter a backtick to escape the JavaScript function

The following payload would throw prompt(): `prompt()\*/ -> (/\* is a comment in JS this prevents the JavaScript function from breaking!)

* Never use alert -> do confirm() or prompt() or something else alert() is filtered far too often!

Were pretty familiar with HTML context but just for review

* Locate sink (reflected input value)
* Break out of the HTML tag that our input is reflected in
* Start testing out payloads, filter evasion …….

This is an example of DOM XSS:

Prompts user to enter their username then sets the inner HTML of the <h1> tag to the value of username variable. We could simply just add a simple <script> payload to throw any XSS vector.

**How to test for XSS (Generally)**

Dependent on Context on but just start by looking for reflection and determining context or testing every possible parameter both are great strategies.

* JavaScript: ‘,”, (,), ` …….
  + See how the app handles ‘, “, ` (backtick) and () these are all useful in breaking out of JS
* HTML: <img src=x>
  + First find HTML injection then test for XSS (create payloads according to context)
* HTML attribute: ‘>”<` …….
  + Backtick , “, ‘ < > anything used to break out of HTML

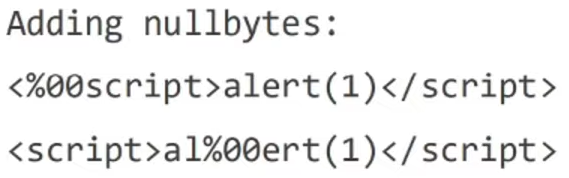
All simple things if it breaks or changes the page… look deeper!!

**Passive Attack Strats:**

* Enter attack vectors in every single input you see during registration.
* Watch for xss popups down the line, this could be broken HTML or a broken image depending on your payload. If you do this this looks deeper into that sink.

This method is low effort but not the most effective. Also, easy to test for hidden features

**Basic Payload Modifications:**

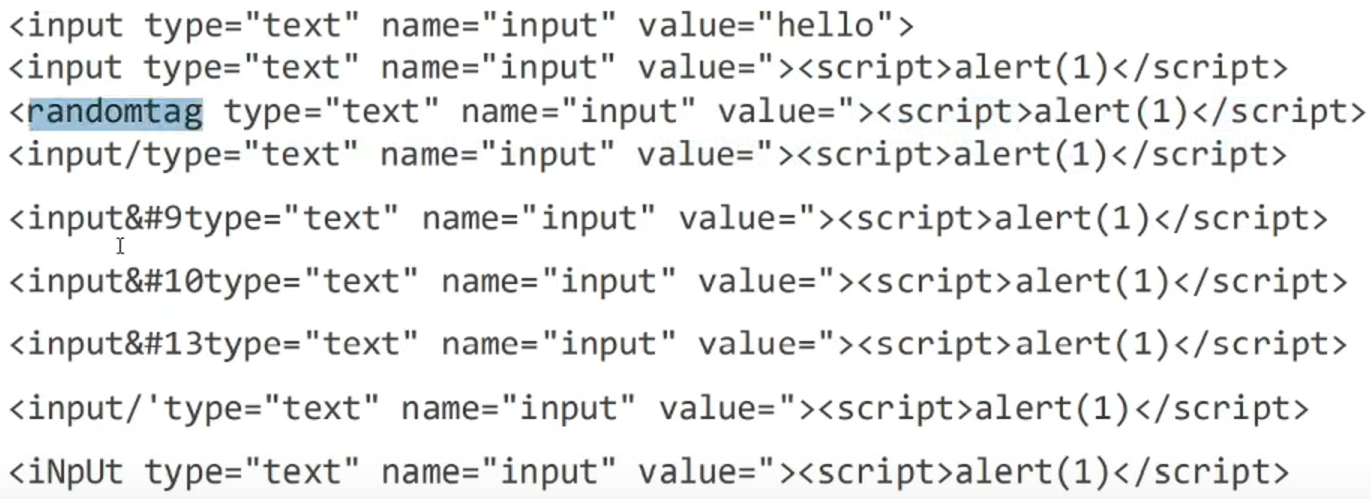
Text

Description automatically generated

Everything you see above is fair game and good to try, random spaces, encoded tabs/ newlines adding null bytes etc…

------------------------------------------------------------------------

**Attribute and Tag Modifications:**



Removing spaces, adding spaces, replacing spaces with /, encoding the /, putting the <Script> after a benign HTML tag, uppercase ………

Text, letter

Description automatically generated Also try to insert null bytes.

------------------------------------------------------------------------------------------------------------------------------------

Event Handlers:

Look at the Burp XSS cheat sheet for all possible event handlers -> just use a fuzzer to find which ones get through the filter.

* Do this when you have already identified a sink and are beginning to test for XSS

------------------------------------------------------------------------------------------------------------------------------------------

**More REALLY COOL payload mods:**

Text

Description automatically generatedaka double tags << >> also try to URL encode the double tags.

----------------------------------------------------------------

Text

Description automatically generated

If common functions are blocked we can try to call the eval() function and pass it an obfuscated version of alert, also try with confirm or prompt!!!!!!! (for the last one you can look up the char code using ASCII (I think) this example above evals to alert(1)

Text

Description automatically generated

Insert a filtered word within that same filtered word sort of tests to see if their filter works recursively in a sense.