**Testing for Reflected XSS:**

XSStrike is great tool which can test for DOM and reflected XSS it can also fuzz and crawl.

* See bookmark in web app resources

Requirements/ What is Reflected XSS:

* Attack must be able to inject browser executable code in a **single** HTTP response.
* This input must not be stored within the application.
* Not persistent!
* Only affects users who clicks on malicious links.
* Attack vector only belong to the URI or HTTP parameters.
* XSS happens from user input that is not processed properly (unsensitized) and then gets executed in client side browsers.

Most common type of XSS!

To hack someone using reflected xss we would need to of course locate xss somewhere then social engineer someone to click that link causing the arbitrary code to get executed in their browser.

First we need to look for sinks. A sink is a place where user input is reflected into the webpage. We need to determine the context of this sink next, meaning is it the sink in HTML, JavaScript …..

* When testing for sinks use a string that wont get picked up by filters and then search using dev tools or search the response (if using zap or burp) for your specific input

The biggest challenge is figuring out which payload to use, and how to encode/ obfuscate it to bypass the WAF or filter.

**Testing Special Chars HTML context:**

HTML Context: for when there is an HTML sink. Again we are looking for direct reflection, no encoding or filtering.

* In all attack vectors test for key **HTML** entities
  + These include: <, >, &, ‘, “, `(backtick) and more
  + Check how the application handles these input and base subsequent attempts off this.

<https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references> big list of entity references!

HTML encoding is a great way to bypass filters in HTML sinks !

* %3c = < | %3e = >

**Testing Special Chars: JavaScript context:**

With javaScript context we need to break out of the function first then insert our code, we can do this with (), “, ‘, ` or other ways. After breaking out we can add our own arbitrary code

Check to see how the application handles the following chars:

-> \n, \r, ‘, “, \, \uXXXX(Unicode values) , `(backtick (Unicode values)

Each of these chars can help us break out of the JavaScript context or craft an attack string based off them. Again these are juts the common/ most important ones but there are some others.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#using_special_characters_in_strings>

check out this link for some more important JavaScript values

**How to test – blackbox:**

This style of testing involves detecting input vectors and testing all user-controlled variables and parameters. this includes less obvious inputs such as: HTTP parameters, POST parameters, POST data, Hidden Fields, predefined radio or selection values.

**->** Analyzing Input Vectors:

* Try specifically crafted input vectors for every single parameter!
  + This means escaping quotes or attributes.
  + And finding the context of the input vector, this means where the input is reflected, is it reflected in JavaScript or a HTML tag/ attribute -> this context ultimately determines which payload we use.

Try to use your imagination and try many different things e.g., encoding.

Check out <https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html> for many beautiful payloads.

-> Checking Impact

If any of our attack vectors from the previous step “catch” or reflect we have to analyze the impact **realistically**.

* Look where the value is reflected.
* Outline which special characters are not properly encoded eg <,>,= …
* Base your subsequent attempts off these findings.
* Blacklist based filters are highly exploitable!

We need to try to exploit of gain information from this, we cannot just throw up an alert and submit it. We must steal cookies or something to show proof of concept/ impact. If not then move on and try to find impact elsewhere. (chances are if you find a payload that works in one place It may work elsewhere. Use the same or similar payloads just look it new places to try to find actual exploitable impact.

**Tools:**

* XSStrike – xss tester, fuzzer, crawler
* PHP Charset Encoder(PCE) – allows us to encode arbitrary text into 65 different types of char sets (encodings) that we can use to customize and obfuscate our payloads
* Hackvertor is an online tool that allows many types of encoding and obfuscation (also a burp extension)

See the enocders bookmark in /hacking/XSS folder

Maybe try out ratproxy and xssproxy too