

# 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](#)[Part-6-information-Gathering-with-FOCA.html](#)[Part-7-information-Gathering-with-Metagoofil.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](#)

## Basics of Javascript for XSS - final

Hello again, In the previous part of this tutorial we learnt about:

1. Various Important Properties and Methods of window & document objects
2. Functions in JavaScript

So in this tutorial, we are going to learn about Event Handlers in JS

Event Handlers in Layman's language are some special attributes used with specific tags (so all tags) use to handle an event or to trigger a specific set of code when an event occurs. Now include inline Javascript without <script> tags but it only executes when an event occurs. Now anything like clicking something, hovering over some tag, double clicking, pressing a key, unfocusing it, loading of a page etc.

A Basic Syntax for that is:

```
<tagname someattribute1=value onSomeEvent="var x=10;alert(x);//javascript code here">
```

A very basic example of Event which executes on-a-click, the Event Attribute name for click example: [Click Me](#)

Now When Someone click on Hyperlink "Click Me" the onclick event triggers and javascript popup and then its redirected.

Some Common Event Handlers:

1. **onmouseover/onclick** : onmouseover occurs when an object with this event is hovered triggered when an object is clicked
2. **onmousedown/onmouseup** : clicked and not left/mouse click is left
3. **onfocus** : typically used with input tag when we focus or select the input field
4. **onblur** : used when we unfocus a field
5. **onkeypress/onkeyup/onkeydown** : onkeypress occurs when a key is pressed, onkeyup or onkeydown occurs when key is pressed and not left.
6. **onload** : occurs when a page loads
7. **onunload** : when someone leaves the page
8. **onerror** : when an error occurs while loading a file

Now there's a way too big lists of event handlers available in your browsers and its not possible but there is a way to get to know about available Event Handlers is open Dev Console (Ctrl+Shift+J) and console will show you (press Ctrl+Space if it didn't show) a long list of available events could just pick any one and Google about them i.e. which event they handle and with which used.

**NOTE:** Since This Post is not regarding XSS but Javascript's Basic Knowledge Required for understanding about those important event handlers which are usually not blocked and could be bypassed by WAFs (WAF) Which we are leaving for one of the upcoming posts of this Series.

XPATH-Error-Based-Injection-  
 Extractvalue.html  
 XPATH-Error-Based-Injection-  
 UpdateXML.html  
 Error-Based-Injection-  
 Subquery-Injection.html  
 sql-evil-twin-injection.html  
 Blind-SQL-Injection.html  
 bypass-login-using-sql-  
 injection.html  
 dump-database-from-login-  
 form-sqli.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  
 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

There is also an event handlers list posted by Dr.Mario(@0x6d61726966f) <http://pastebin.cc> which comes handy when bypassing WAFs filtering some common above listed Handlers.

So This is all for Event Handlers just wanted to introduce you guys with event handlers, lets move on to AJAX Requests.

**AJAX** -ie. "Asynchronous Javascript And XML", which can be used to :

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

**Steps to Create a Very Basic AJAX Request:**

1. Create an XMLHttpRequest(also called XHR) Object
2. Use "onload" method on that object to define what function to call after the request has been received
3. Use "open" method on that object to define Request Method(GET/POST/PUT/PATCH/DELETE/External File to make Request to.
4. Used "send" method to Make that request.

so a Basic Structure to Make an AJAX Request looks like this:

```

<script>
var xhrobj=new XMLHttpRequest();//creates XHR object
xhrobj.onload=function()
{

if(this.status===200) //Used to check if the Response Code is 200 OK ie. Page loaded fine or not
{
//some things to do after Response has been Successful
}

};
xhrobj.open("GET","URL/file to load");
xhrobj.send();//if method is POST , POST data is to be sent as its parameter/argument like this:
</script>
  
```

**NOTE :** This won't work if u just open this file in "Chrome" locally ie. using file:// protocol How does it work? It allows that. So either open in Firefox or set up a Web server and do on that.

### **SAME ORIGIN POLICY (SOP)**

Now you can't Just read responses from any Web Page out there, if that was so then, anyone could make a Request to Your Bank/Gmail and Read Responses ie. Read your sensitive data. So this is POLICY(SOP) comes into play, it restricts which "origin" is allowed to read responses from must be Same "origin" if we want to read server responses. ("origin" is explained below)

To circumvent SOP, we have to follow some rules:

Two pages have the "same origin" if

- Protocol (<http://>,<ftp://> etc)
- Specified Port(<http://host.tld:PORT/>)
- host (<http://host.tld/>)

are the same for both pages.

For More Details refer to: <https://developer.mozilla.org/en-US/docs/Web/Security/Same-Origin-Policy>

### **Problem With SOP**

But this is a problem when we want to read Responses from another origin, so in that Case, (CORS - Cross Origin Resource Sharing) comes into action. It is used to allow reading server responses from "other origin" in that case we are able to read responses from External Origin without SOP coming to play.

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

When a CORS request is made a "Origin" Header is also sent along Which contains the origin and in Response of HTTP Header comes like,

```
Access-Control-Allow-Origin: http://host
```

which means response could only be read/accessed by <http://host> origin

Now This causes another problem, Since sometimes we have this header's value set \* (which host) which allows to read sensitive server responses. But by default cookies are not sent again,

```
Access-Control-Allow-Credentials: true; // its another header returned in response which is
```

However, Browser Implements their own Security by not allowing below headers together:

```
Access-Control-Allow-Origin: *  
Access-Control-Allow-Credentials: true
```

So We are Safe again if the "Origin" Header is Validated Properly if its not , It leads to Sensitive Server

For More Details on This Issue: <http://www.geekboy.ninja/blog/exploiting-misconfiguration-resource-sharing/>

For More Details on CORS: [https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_)

#### Reading Server Response:

Below is the javascript that can be used to read contents from readme.html file

```
<script>  
var xhrobj=new XMLHttpRequest();  
xhrobj.onload=function()  
{  
  
if(this.status===200)  
{  
var resp=this.responseText;//this.responseText contains the server response, there's also re  
alert("Server Response: "+resp);//this will alert the HTML content of server response  
}  
  
};  
xhrobj.open("GET","/readme.html");  
xhrobj.send();  
</script>
```

AJAX Request would be required in Scenarios Where we will be stealing nonces(CSRF Performing CSRF Actions on behalf of other Users.

So I Think This Should be the end of the "Booooooring" Javascript Basics for XSS though :P we few more basics if required in future. Finally, :D we could start with XSS from the next Post.

Author : Rahul Maini

Date : 2017-05-13