Tutorials Browser

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(sometimes works with all tags) use to handle an event or to trigger a specific set of code when an event occurs. They are used to include inline Javascript without <script> tags but it only executes when an event occur. Now that event could be anything like clicking something, hovering over some tag, double clicking, pressing a key, focusing an element, 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 event is "onclick", for example: Click Me

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

Some Common Event Handlers:

1. **onmouseover/onclick** : onmouseover occurs when an object with this event is hovered while onclick is 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 occurs when key is left and 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 to write about all, but there is a Way to get to know about available Event Handlers is open Dev Console (Ctrl+Shift+J) and type "on" and console will show you(press Ctrl+Space if it didn't show) a long list of available event handlers. Now You could just pick any one and Google about them ie. which event they handle and with which tags they could be used.

NOTE: Since This Post is not regarding XSS but Javascript's Basic Knowledge Required for XSS, I haven't written about those important event handlers which are usually not blocked and could be used to bypass XSS filters(WAF) Which we are leaving for one of the upcoming post of this Series.

There is also an event handlers list posted by Dr.Mario(@0x6d6172696f) http://pastebin.com/raw/WwcBmz5J, 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 come to next topic ie. 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 made
3. Use "open" method on that object to define Request Method(GET/POST/PUT/PATCH/DELETE etc), URL or 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 Successfully.
{
//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 xhrobj.send("nam
</script>
```

NOTE : This won't work if u just open this file in "Chrome" locally ie. using file:// protocol However "Mozilla Firefox" 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 an AJAX Request to Your Bank/Gmail and Read Responses ie. Read your sensitive data. So this is where SAME ORIGIN POLICY(SOP) comes into play, it restricts which "origin" is allowed to read responses from which "origin". There must be Same "origin" if we want to read server responses. ("origin" is explained below)

To circumvent SOP, we have to have 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 "whitelisted" Origins. In that case we are able to read responses from External Origin without SOP coming to play.

When a CORS request is made a "Origin" Header is also sent along Which contains the origin of the AJAX Request 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 means it allows any host) which allows to read sensitive server responses. But by default cookies are not sent so we are safe but again,

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

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 Disclosure from Server
For More Details on This Issue: http://www.geekboy.ninja/blog/exploiting-misconfigured-cors-cross-origin-resource-sharing/
For More Details on CORS: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

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 responseXML pro
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 Tokens) via XSS and Performing CSRF Actions on behalf of other Users.

So I Think This Should be the end of the "Boooooring" Javascript Basics for XSS though :P we will keep covering a 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