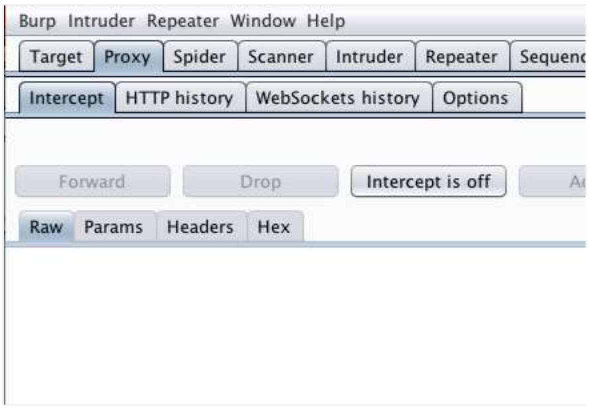
Exploiting XSS - Injecting into Direct HTML

For the purposes of detecting XSS, Direct or Plain HTML refers to any aspect of the HTML response that is not a tag attribute or scriptable context. This will demonstrate how to identify reflections of user input, and inject an XSS attack in to such a context.

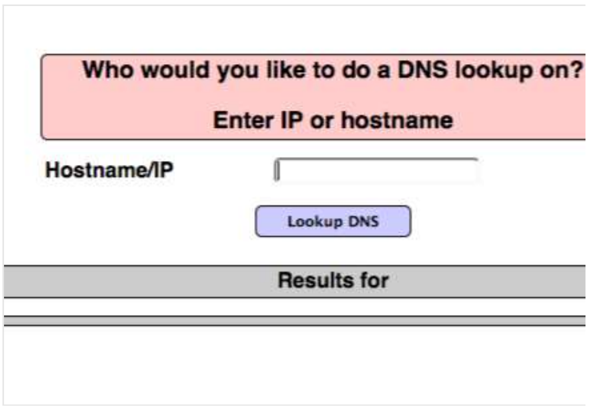
The example uses a version of "Mutillidae" taken from OWASP's Broken Web Application Project. Find out how to download, install and use this project.

First, ensure that Burp is correctly configured with your browser.

With intercept turned off in the Proxy "Intercept" tab, visit the web application you are testing in your browser.

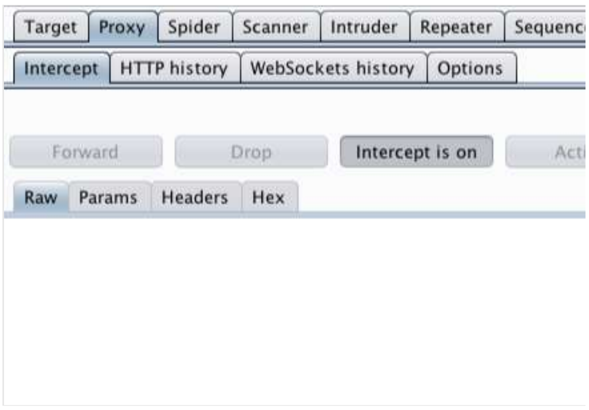

Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequenc
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is off
Raw Params Headers Hex

Visit the page of the website you wish to test for XSS vulnerabilities.



Return to Burp.

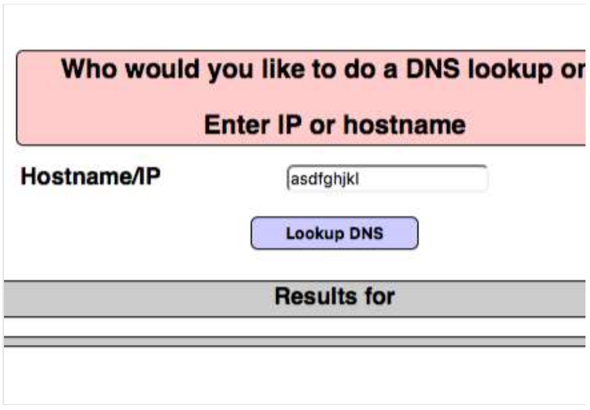
In the Proxy "Intercept" tab, ensure "Intercept is on".


Target Proxy Spider Scanner Intruder Repeater Sequenc
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on
Raw Params Headers Hex

Enter some appropriate input in to the web application and submit the request.

The first stage in the testing process is to submit a benign string to each entry point and to identify every location in the response where the string is reflected.

Choose an arbitrary string that does not appear anywhere within the application and that only contains alphabetic characters and therefore is unlikely to be affected by any XSS-specific filters.



Who would you like to do a DNS lookup or

Enter IP or hostname

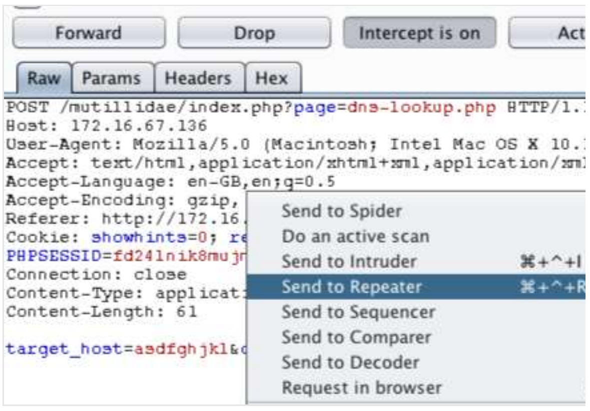
Hostname/IP	asdfghjkl

The request will be captured by Burp. You can view the HTTP request in the Proxy "Intercept" tab.

You can also locate the relevant request in various Burp tabs without having to use the intercept function, e.g. requests are logged and detailed in the "HTTP history" tab within the "Proxy" tab.

Right click anywhere on the request to bring up the context menu.

Click "Send to Repeater"



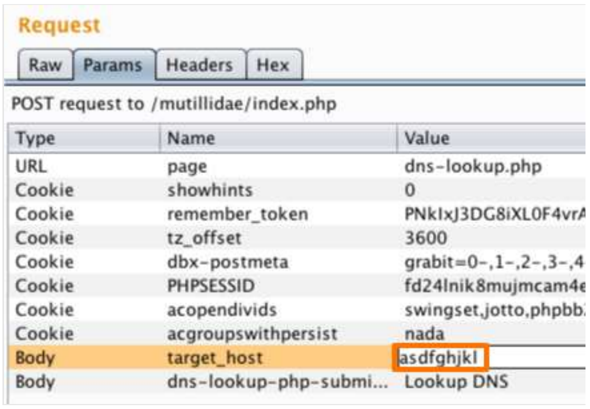
```
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.
Accept: text/html,application/xhtml+xml,application/xm
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip,
Referer: http://172.16.
Cookie: showhints=0; re
PHPSESSID=fd24lnik8mujr
Connection: close
Content-Type: applicat
Content-Length: 61

target_host=asdfghjkl&d
```

Go to the "Repeater" tab.

Here we can input various XSS payloads in to the input field of a web application.

We can test various inputs by editing the "Value" of the appropriate parameter in the "Raw" or "Params" tabs.

Submit this string as every parameter to every page, targeting only one parameter at a time.



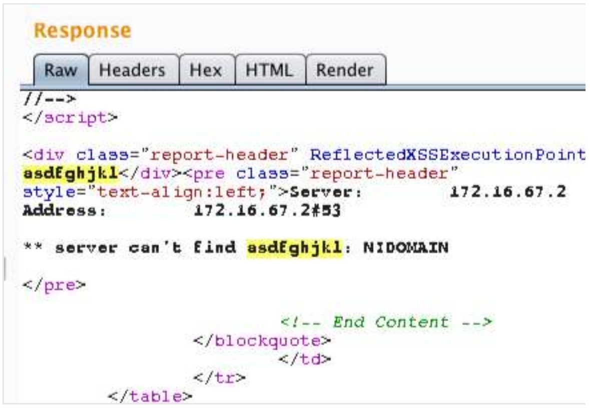
POST request to /mutillidae/index.php

Type	Name	Value
URL	page	dns-lookup.php
Cookie	showhints	0
Cookie	remember_token	PNkIxJ3DG8iXL0F4vrA
Cookie	tz_offset	3600
Cookie	dbx-postmeta	grabit=0-,1-,2-,3-,4
Cookie	PHPSESSID	fd24lnik8mujmcam4e
Cookie	acopendivids	swingset,jotto,phpbb
Cookie	acgroupswithpersist	nada
Body	target_host	asdfghjkl
Body	dns-lookup-php-submi...	Lookup DNS

Review the HTML source to identify the location(s) where your unique string is being reflected.

If the string appears more than once, each occurrence needs to be treated as a separate potential vulnerability and investigated individually.

Determine, from the location within the HTML of the user-controllable string, how you need to modify it to cause execution of arbitrary JavaScript.

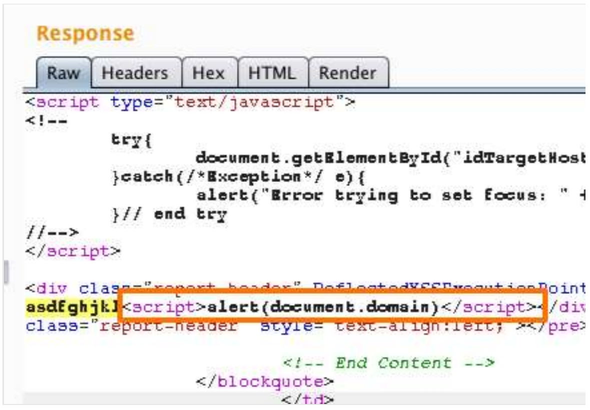


```
//-->
</script>

<div class="report-header" ReflectedXSSExecutionPoint
asdfghjkl</div><pre class="report-header"
style="text-align:left;">Server:        172.16.67.2
Address:        172.16.67.2#53

** server can't find asdfghjkl: NXDOMAIN

</pre>

                        <!-- End Content -->
                    </blockquote>
                        </td>
                    </tr>
            </table>
```

The process of crafting an XSS exploit is often one of trial and error. One must consider how to introduce JavaScript without causing an error and work around any defensive filters.

Test your exploit by submitting it to the application. If your crafted string is returned unmodified, the application is vulnerable.

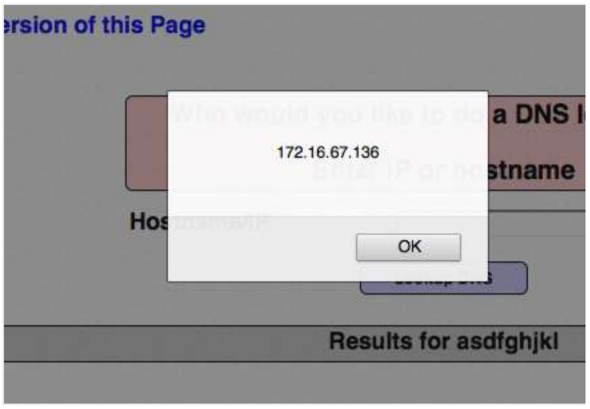
In this example, we can open up a <script> tag to introduce our JavaScript.



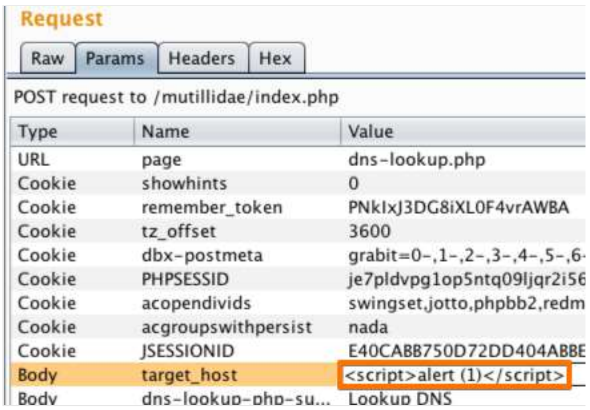
```
<script type="text/javascript">
<!--
        try{
                document.getElementById("idTargetHost
        }catch(/*Exception*/ e){
                alert("Error trying to set focus: " +
        }// end try
//-->
</script>

<div class="report-header" ReflectedXSSExecutionPoint
asdfghjkl<script>alert(document.domain)</script></div
class="report-header" style="text-align:left;"></pre>

                        <!-- End Content -->
                </blockquote>
                        </td>
```

Double-check that your syntax is correct by using a proof-of-concept script to display an alert dialog.

Confirm that this appears in your browser when the response is rendered.

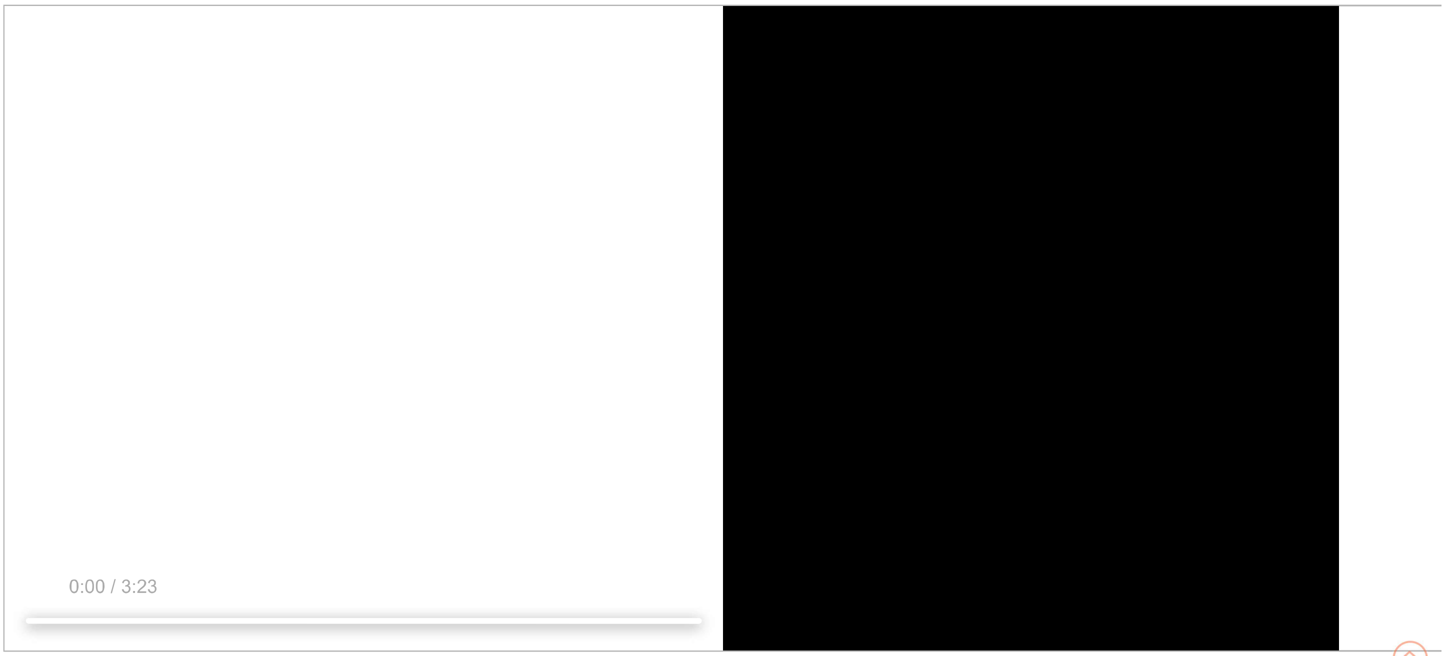


Note: In any cases where XSS was found in a POST request, you can use the "change request method" option in Burp to determine whether the same attack could be performed as a GET request.



POST request to /mutillidae/index.php

Type	Name	Value
URL	page	dns-lookup.php
Cookie	showhints	0
Cookie	remember_token	PNkIxJ3DG8iXL0F4vrAWBA
Cookie	tz_offset	3600
Cookie	dbx-postmeta	grabit=0-,1-,2-,3-,4-,5-,6-
Cookie	PHPSESSID	je7pldvpg1op5ntq09ljqr2i56
Cookie	acopendivids	swingset,jotto,phpbb2,redm
Cookie	acgroupswithpersist	nada
Cookie	JSESSIONID	E40CABB750D72DD404ABBE
Body	target_host	<script>alert (1)</script>
Body	dns-lookup-php-su...	Lookup DNS



Related articles:

Getting started with Burp Proxy

Using Burp Repeater