HTTP Response Splitting Attack | InfoSec Institute – IT Training and Information Security Resources

http://resources.infosecinstitute.com/http-response-splitting-attack/ October 15, 2011

Introduction:

In this paper we will discuss HTTP Response Splitting and how the attack can actually be carried out. When we're clear about how it works, because it is an often misunderstood topic, we'll then look at how Response Splitting can be used to carry out Cross Site Scripting(XSS). We'll then discuss if its possible to perform a CSRF attack if the site is vulnerable to Response Splitting. Finally, we'll look at the mitigations for each of these attacks. If you think this interests you, read on!

What is HTTP Response Splitting?

Think of a page which can be displayed in multiple languages. The page by default is displayed in English but with an option to select another language from a dropdown and have the page displayed in that instead. Lets say the request for the initial page results in a 302 redirect to http://www.abc.com/index.php?lang=en. A user from Germany though, wants to display the page in German instead and selects that option from the list of available languages. This results in a 302 redirect being sent for the German page to the server – http://www.abc.com/index.php?lang=german. The user's browser will follow the redirect and display the German page to the user.

Lets now think of the main parts of the HTTP 302 redirect response. This is what it will look like:

HTTP/1.1 302 Moved Temporarily

Location: http://www.abc.com/index.php?lang=en

OR

HTTP/1.1 302 Moved Temporarily

Location: http://www.abc.com/index.php?lang=german

You'll quickly notice that the only thing that has changed is the value of the lang parameter. This hence means that this value is controlled by the user and he can set it to absolutely anything he wants. It is precisely this property that an attacker targets using HTTP Response Splitting.

Instead of supplying just 'german' as a value he will instead supply a value which contains the following:

> a) The value 'german'
> b) CR/LF – %0d%0a
> c) A response with Content Length 0 [Because it really does not matter what this contains]
> d) CR/LF – %0d%0a
> e) A response which contains malicious content [For e.g Javascript which will download malware when the page is visited]

Lets look at **c)** – The first response. The way the HTTP protocol works is – 1 request : 1 response. That's what this first response is. Its just a crafted response to the first request that was sent; namely http://www.abc.com/index.php?lang=german. Since we're not really worried about this response and its content, we just set Content-Length: 0 in the response header.

The CR/LF is a delimiter between responses. So if we put a CR/LF as in **d)** and start our 2nd response it is valid as per the HTTP protocol and will be processed. You can put pretty much anything in this response. So for e.g if we just want to display a message “Hello, you have been phished”, we can do just that. This response would then look as follows:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 41
Hello, you have been phished
```

Unclear? Lets summarize once. Its the attacker who is controlling a parameter and sending 1 request along with 2 responses; both set by him, to the server. The first request, which was one for the German page is mapped to the first response, while the 2nd response (as of now) is unaccounted for, it’s just hanging – as there is no request to which it can be mapped to. Remember, HTTP needs a response (with whatever error code) but it needs a response for every request that is sent. So a hanging HTTP response just ..does not work.

Now read carefully…because this is where most people (myself included in the past) have got stuck. Inorder to address the second hanging response, the attacker quickly sends a request for a valid publicly accessible(usually) page on the server, like branches.html.

So he sends a request like:

```
GET /branches.html HTTP/1.1
Host: www.abc.com
```

…. immediately after he sends the first request which contained the ‘fully controllable’ parameter. THIS is the second request to which the second ‘**Hello, you have been phished’ page** maps to. 2 requests… 2 responses. Clear?

No, I guess not…while you might have understood how mapping happens, it is still not clear how this attack affects anyone else. After all, the attacker is doing all this on his own machine, and modifying requests only for himself.. so if at all, its only he who is affected…not any one else. And really, why would the attacker attack himself? It just doesn’t make any sense. **That is where the absolute necessity of a proxy server or some device which sits inbetween and caches requests and responses comes in.**

The attacker must be behind a proxy server which then passes his requests to the server on the Internet. If he wants to infect other users, all these users must also be behind the same proxy server. So lets resummarize again now:-

a) Attacker sends a request which contains a value and 2 responses, separated by %0d%0a. The full request for the example will be as follows:

```
http://www.abc.com/index.php?lang=german%0d%0aContent-
Length:%200%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2041%0d%0aHello, you have been phished
```

b) This request is sent to www.abc.com but…but crucially, it passes through the intermediate proxy server. Now on the proxy server, the first request is mapped to the first response. The second response hangs very briefly as there is no matching request.

c) Immediately after sending the first request the attacker sends a new request(2nd request) to the site (and hence through the proxy server as well) which is:

```
GET /branches.html HTTP/1.1
Host: www.abc.com
```

d) The moment the proxy sees the request for branches.html it maps it to the second response (“You have been phished”). **So a future request for branches.html will NOT display the list of the bank’s branches but the malicious page instead. Yes, for everyone. Not just the attacker.** Why? Because that’s what a caching proxy server does.. caches responses for requests often made. So if a request for branches.html always produced the same static list of branches, the proxy server is almost certainly going to cache the response to this request. It will return this cached response, the next time a request is made for branches.html. The attacker though, now, has poisoned the proxy’s cache and forced it to display his malicious response instead of the static branches list….until the cache expires.

I do hope that was clear. If not read through the article, till here, slowly, one more time. The key point really, is the fact that its the attacker who sends the second request to force the proxy to store the 2nd request – 2nd response mapping. Once that is understood, the concept is clearer.

Cross Site scripting – Through response splitting

I’d like to point out here that I’m not explaining Cross Site Scripting and its types in great detail over here. There are plenty of great articles available online (I’ll link a few in the References section) that you can read to gain further clarity on the same.

Now that we’re reasonably clear about what Response Splitting is, can we take this one step forward from an attacker’s perspective. Can we, through response splitting, run Javascript on a victim machine and try and eventually gain complete control of his browser? Yes, we can, by just extending the example that we took earlier a little bit further..

If you remember, the second malicious response that we’d crafted above was a simple HTML page which said that the user had been phished. Instead of the simple HTML page, we would have to write some Javascript code instead. Instead of a simple HTML page that gets displayed, the Javascript would run instead on the user’s browser. Control that the attacker will have over the user’s browser will depend completely on the Javascript code that was written. So for e.g .. Extending the same example which we discussed earlier in this article, the URL that the attacker would craft would be something like this:

http://www.abc.com/index.php?lang=german%0d%0aContent-Length:%200%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent- Length:%20%0d%0aalert(‘Running JS on your machine’)

He would then send a request for branches.html like before and it would get mapped to this second response which contains malicious Javascript. The proxy cache will get poisoned like before and when a user who sits behind this proxy accesses branch.html – the JS will run on his machine.

In case there is a parameter that is vulnerable to XSS on the target site, the same logic can be used to exploit that as well. Only, in this case there’ll be a script in the vulnerable parameter, which is part of a request and not Javascript in the body of the HTML page as above.

The Javascript in this example is extremely straightforward and will only pop up a little alert box. However there can be more complex JS that can be written which can result in an attacker gaining complete control of a user’s browser and eventually, control of the user’s machine as well. An exploitation framework called BeeF is of great help to the attacker here, if he wants to do this. He will just have to write a little Javascript and point it to the BeeF controller which will be installed on some machine which is reachable from the victim.

Cross Site Request Forgery – Through Response Splitting

Again, I’m not diving very deep into how the CSRF attack works. You can take a look at the references I provide for further details about the same. In a nutshell though, a person who is a victim to a CSRF attack, will perform a ‘non-view’ operation unknowingly.Note that this is an operation that he wouldn’t have otherwise performed.

The pre-requisites for a CSRF attack to happen are that the victim needs to be logged in to the site for which the oepration is to be performed. So if the operation that the attacker wants to trick the user into performing is a ‘Delete my Google Profile’, the user will have to be logged into Google’s systems, when the operation is performed. Secondly, the attacker must be able to predict the exact structure of the request including the parameter values. So for e.g A fund transfer request would look like GET /transfer/php?acc1=1000&acc2=2000&amt=900. If a user sends a GET request like mentioned, while being logged in to www.abc.com , a fund transfer will automatically happen from acc1 to acc2.

In a CSRF attack, the attacker somehow tricks the user to click that link using Social Engineering techniques or gets the user to visit a page under the attacker’s control, where this request is performed in the background, transparent to the user.

Now, back to Response Splitting, you’ll remember that the attacker used Response Splitting to poison the page branches.html and insert malicious Javascript into the page to attempt to run scripts on the user’s browser. In the case of CSRF you will need to ensure that the action (fund transfer URL) is automatically performed when the user accesses the poisoned page. So in other words, branches.html can contain a small image which automatically loads each time the page loads. The <IMG SRC> tag of this image though will send a request – /transfer/php?acc1=1000&acc2=2000&amt=900 to the www.abc.com server.

So the entire malicious URL, while performing response splitting will be as follows:

```
http://www.abc.com/index.php?lang=german%0d%0aContent-
Length:%200%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent- Length:%20%0d%0a< / body>< / html>
```

So whenever the user behind the proxy visits the poisoned page(branches.html) and he’s signed in to www.abc.com in another tab, the operation will succeed in the background and funds will be transparently transferred to the attacker’s account.

Sample Code:

Lets have a quick look at how some code which is vulnerable to response splitting would actually look like in PHP. For starters we’ll have a small HTML file **(respsplit1.html)**, containing a drop down where you can select the language you want. That will look like this:

```
<HTML>
<BODY>
<FORM NAME=”form” action=”respsplit1.php” method=”GET”>
<select name=”lang”>
<option value=”EN”>English</option>
<option value=”GER”>German</option>
</select>
<INPUT TYPE=”submit” name=”Submit” value=Submit></INPUT>
</FORM>
</BODY>
</HTML>
```

After selecting a language, you click Submit and your input is submitted to a PHP file **respsplit1.php**. All this code does is take your input, use it to create a redirect URL and send you a 302 Redirect response. The code for respsplit1.php is as follows:

```
<?php
$lang = $_GET['lang']
header(“Location: http://localhost/respsplit2.php?lang=$lang”);
?>
```

Open up the HTML file in a browser and trap the request in Burp Proxy or any other proxy editor. Look at the response after you hit Submit on the first page. You'll clearly see that the value that you selected from the dropdown is a part of the Location: header in the 302 response. So if you want to change stuff here, you'll just have to trap the request before it reaches the server and edit the value for the lang parameter with your malicious string (1 request + 2 responses).

respsplit2.php does nothing but print whether the language you selected was English or German

```
$a=$_GET['lang']; if (strcmp($a,'EN') == 0) echo "Language selected is English"; elseif
(strcmp($a,'GER') == 0) echo "Language selected is German"; else echo "No valid language
        selected "; ?>
```

Now when I tried this out my own PHP framework stripped off the %0d%0a characters in the response header, which I tried to pass through as part of input, thus protecting me by default. If however, you are using an older framework which doesn't protect you by default, its very easy to check the code for a carriage return character and not proceed. That filtering function could be like this:

```
<?php
$pattern1 = "/\%0d/";
$pattern2 = "/\%0a/";

$lang = $_GET['lang'];
$r = preg_match($pattern1 , $lang);
$s = preg_match($pattern2 , $lang);

if (($r > 0) || ($s > 0)){
echo 'Carriage Return found in user input';
echo "<BR>";
}
else {
header("Location: http://localhost/respsplit2.php?lang=$lang");
}

?>
```

There can be more efficient filters written, which whitelist just alphabets and numbers and block everything else; but this is just a sample of how defensive code can be used. If you forget to write these filters, you're at the mercy of the framework you use. If, like my current PHP framework (Apt repository – Ubuntu 10.04) there is inbuilt protection..you'll be fine – else you may be vulnerable to an attack.

Many known response splitting vulnerabilities have been found. A link to the same is given in the references section.

Mitigation:

- **Response Splitting:-** Use server side validation and disallow CRLF characters in all requests where user input is reflected in the response header.
- **XSS:** White List and Black List filtering(Input Validation), Escape HTML(Output Validation)
- **CSRF:**Use AntiCsrf tokens so that the attacker cannot predict the exact structure of the request to forge

Conclusion:

A response splitting attack is possible only if there is a proxy server which multiple users use to connect to various websites. The cache of the proxy server is poisoned and the user becomes a victim whenever the proxy cache serves that page. Note though that not all proxy servers are vulnerable to response splitting; the details of this though are beyond the scope of this article. If you’re really interested in knowing more about this attack, I strongly recommend you read Amit Klein’s excellent paper (Reference 1) on the same.

References:

- Response Splitting - http://packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf
- XSS – http://www.technicalinfo.net/papers/CSS.html
- XSS Mitigation – https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sh
- CSRF – https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29
- CSRF Mitigation – https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet
- BeeF Browser Exploitation Framework – http://beefproject.com/
- Learn PHP (Tutorial) – http://www.w3schools.com/php/
- PHP Functions – http://php.net/quickref.php
- Disclosed Response Splitting vulnerabilities – http://cwe.mitre.org/data/definitions/113.html

Incoming search terms: