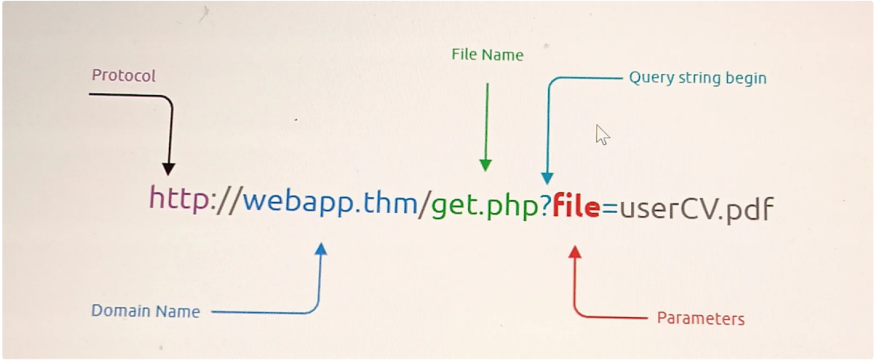
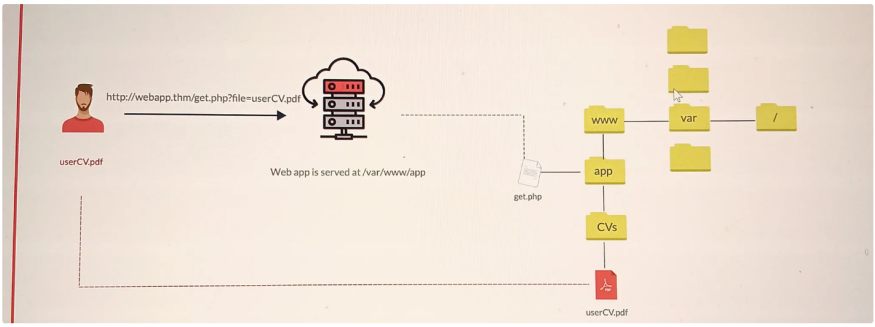
File Inclusion - 1. Local File Inclusion 2.Remote File Inclusion , and directory traversal

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 URL.



First, the user sends as 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 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 happen?

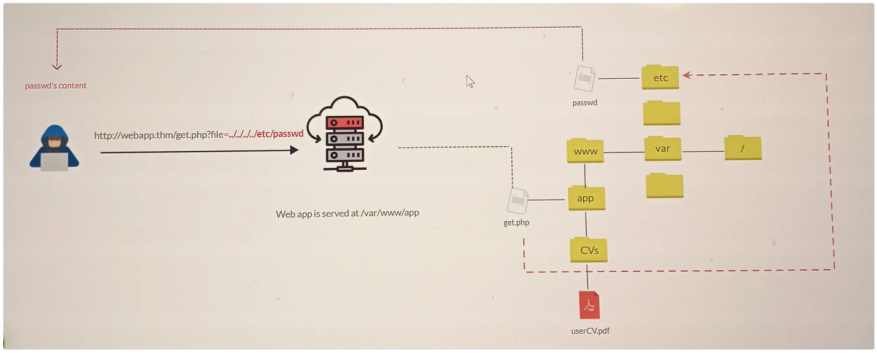
Commonly found and exploited in various programming languages for web applications, such as PHP that are poorly written and implemented. The main issue of these vulnerabilities is the input validation, in which the user inputs are not sanitized of validate, and the user controls them. When the input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

Path Traversal

This vulnerability allows an attacker to read operating system resources, such as local files on the server running an application. The attacker exploit this by manipulating and 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 in passed to a function such as file_get_contents in PHP.

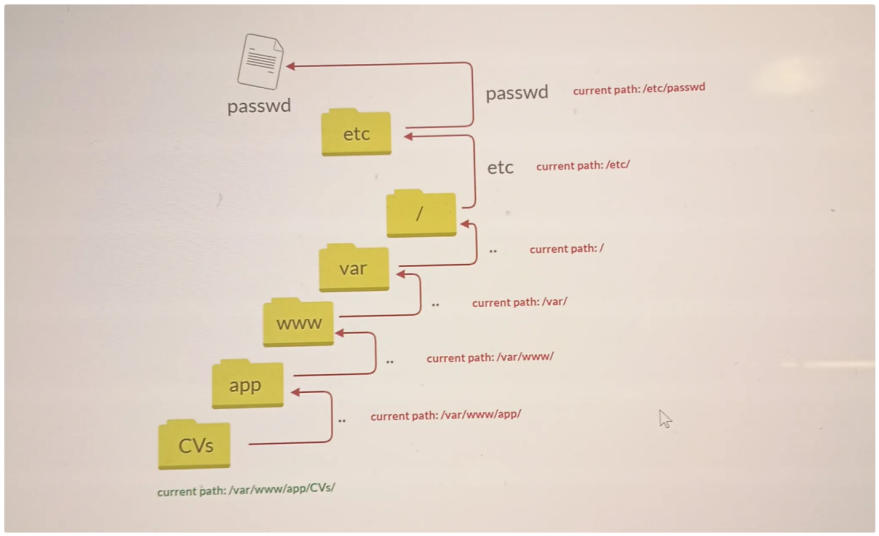
The following graph shows how a web application stores files in /var/www/app. The happy path would be the user requesting the content of userCV.pdf from a defined path /var/www/app/CVs .



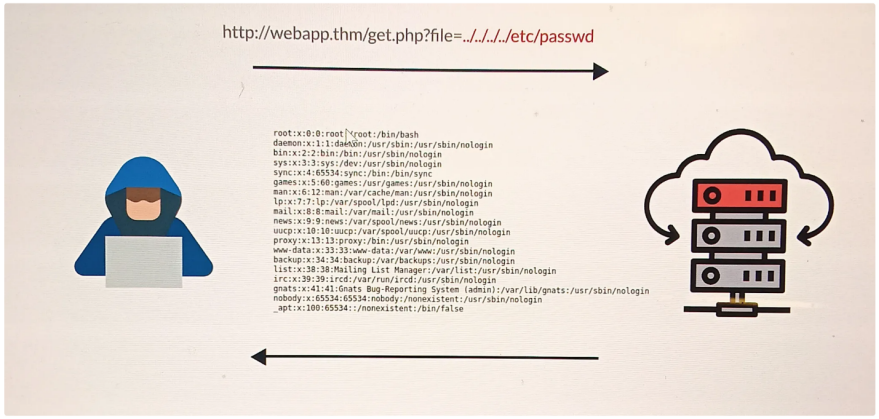
Path Traversal attacks, also known as the ../ attack, take advantage of moving, the directory one step up using the double dots. If the attacker finds the entry point, which in this case get.php?file= , then the attacker may send something as follows,

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 windows server, the attacker needs to provide Windows paths.

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	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

Local file Inclusion

LFI attacks against web applications are often due to a developer's lack of security awareness. With PHP, using functions such as include, require, include_once, and require_once often contribute vulnerable web applications. This time we are using PHP but keep in mind that this attack can occur when using other languages such as ASP, JSP or even Node.js apps.

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 done by sending the following HTTP request as follows: http://webapp.thm/index.php?lang=EN.php to load English page or http://webapp.thm/index.php?lang=AR.php to load Arabic page, where EN.php and AR.php files exist in 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.

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

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

Developer decided to use the include function to call PHP pages the languages directory only via lang parameters.

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

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

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

LFI part 2

1.

```
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 entry. Thus the valid input will be something as follows: index.php?lang=EN , where the file EN is located inside the given languages diretand named EN.php .

Also, 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 directory traversal section, to get out the current folder. Let's try following: http://webapp.thm/index.php?lang=../../../../etc/passwd because we know four levels .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 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. %00 trick is fixed above PHP 5.3.4.

2.In this section, developer decided to filter keywords to avoid disclosing sensitive information. There are 2 possible methods to bypass the filter. First, byusing Null Byte or the current directory trick at the end of the filtered keyword /.

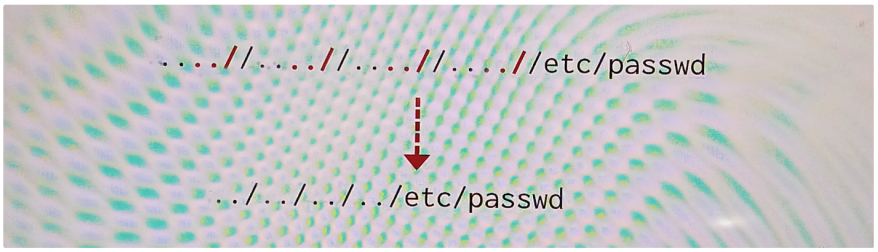
3). Developer starts to use input validation by filtering some keywords. Let's test out and check the error message, and we got error using../ tricks.

```
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, we know that the web application replaces the ../ with the empty string. There are couple of techniques to bypass it.

1. We can send the following payload to bypass it : ....//....//....//....//

This will work 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.



Finally, we'll discuss the case where 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. To exploit this, we need to include the directory in payload.

Remote File Inclusion RFI

It is technique to include remote files 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 need to be on.

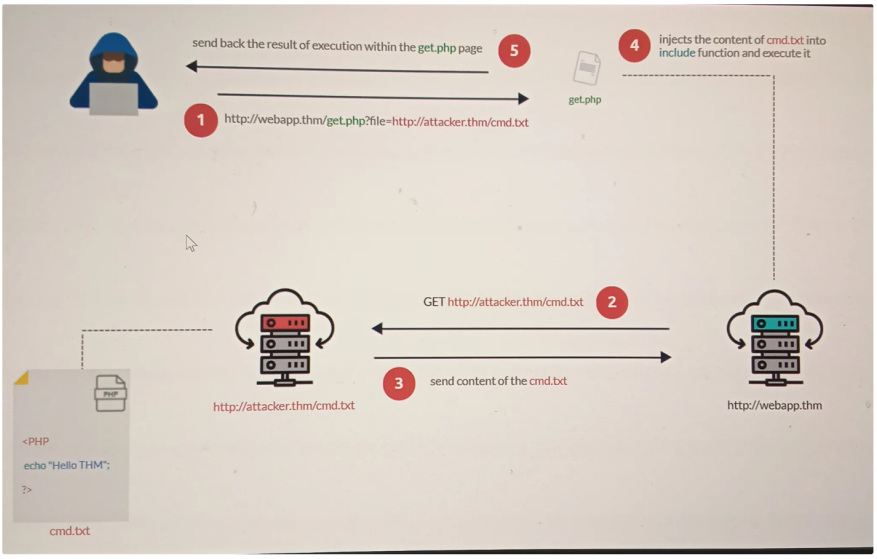
The risk of RFIis higher than LFI since RFI vulnaribilities allow an attacker to gain Remote Command Execution on the server. Other consequences of a successful RFI attack include:

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 vulnarable application server.



RFI steps:

Let's say that the attacker hosts a PHP file on their own server cmd.txt that contains printing message Hello Them.

First attacker injects the malicious URL passes into the include function. Next web app server will send a GET request to the malicious server to fetch the file.

Remediation

As a developer, it's important to be aware of web application vulnerabilities, how to find them, and prevention method. To prevent the file Inclusion vulnaribilities, 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. WAF is good option to help mitigate web application attacks.
4. Disable some PHP features that cause file Inclusion vulnaribilities.
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.