Directory Traversal

(Owasp ref.: Broken Access Control)

A flaw which can lead to directory listing, LFI and RFI and
When exploited can cause loss of sensitive data and even RCE.

Disclaimer:

This is just for an educational purpose and as a method of cyber security awareness. I highly recommend not to harm any institution/Organization without proper permissions and ROE.

Author:

Hey guys, I am vijay reddy. An enthusiastic guy who loves to explore and learn more about cyber security and other fields. I have worked on both INFRA and application security.

Inspiration:

We have many things to learn and knowledge to share.

Art of Thanks:

Thanks to my friends and family.

Bibliography:

Port swigger (https://portswigger.net/)

Kontra (https://application.security/)

Tool: Burp suite

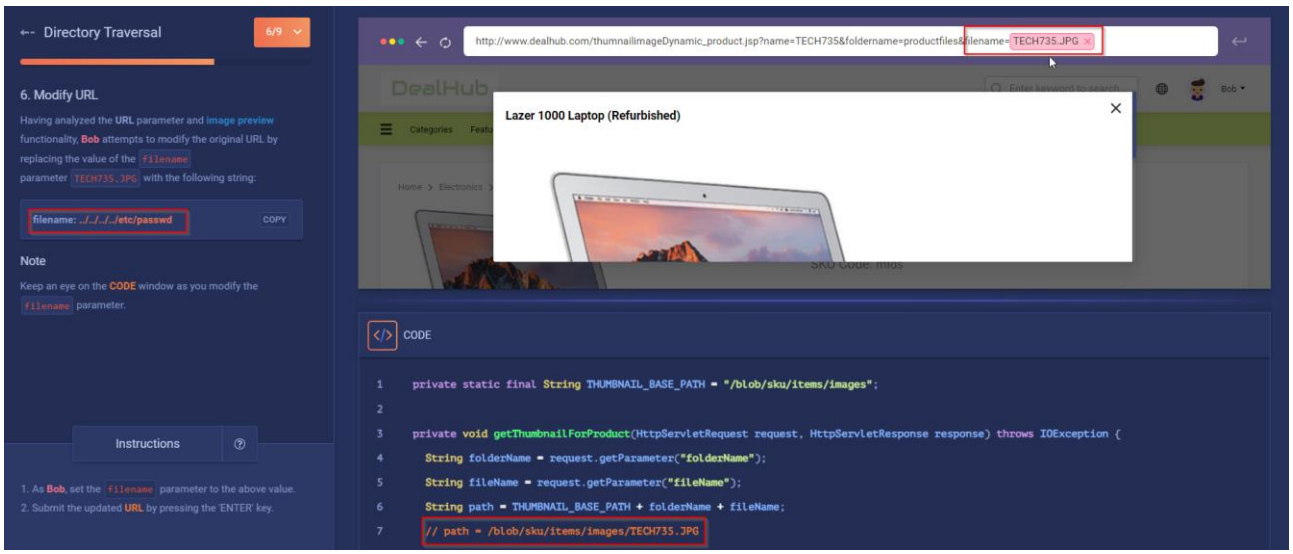
Definition by Kontra:

The ability to get an application to resolve file paths like ../../../../etc/passwd and to display the file's content is known as Directory Traversal attack. It allows the attacker to access and explore the server's file system, possibly extracting sensitive information.

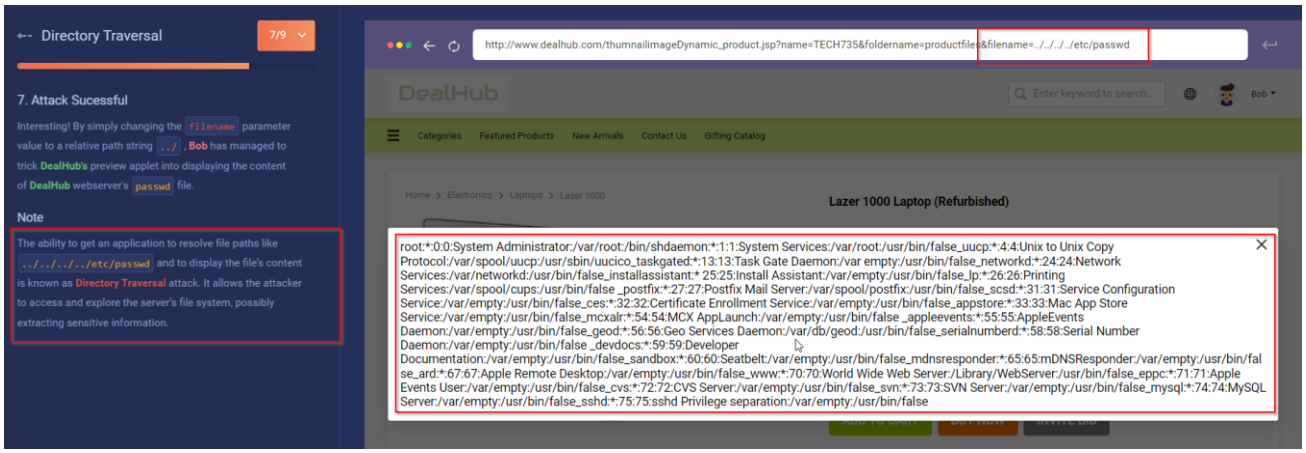
Implemented source code:

```
private void getThumbnailForProduct(HttpServletRequest request, HttpServletResponse response) throws IOException {
  String folderName = request.getParameter("folderName");
  String fileName = request.getParameter("fileName");
  String path = THUMBNAIL_BASE_PATH + folderName + fileName;
```

When browsed to an image the URL path resolves to a file named has TECH735.png which means somewhere this file is stored and is now retrieved when tried to access. This can be considered has a clue for DT.

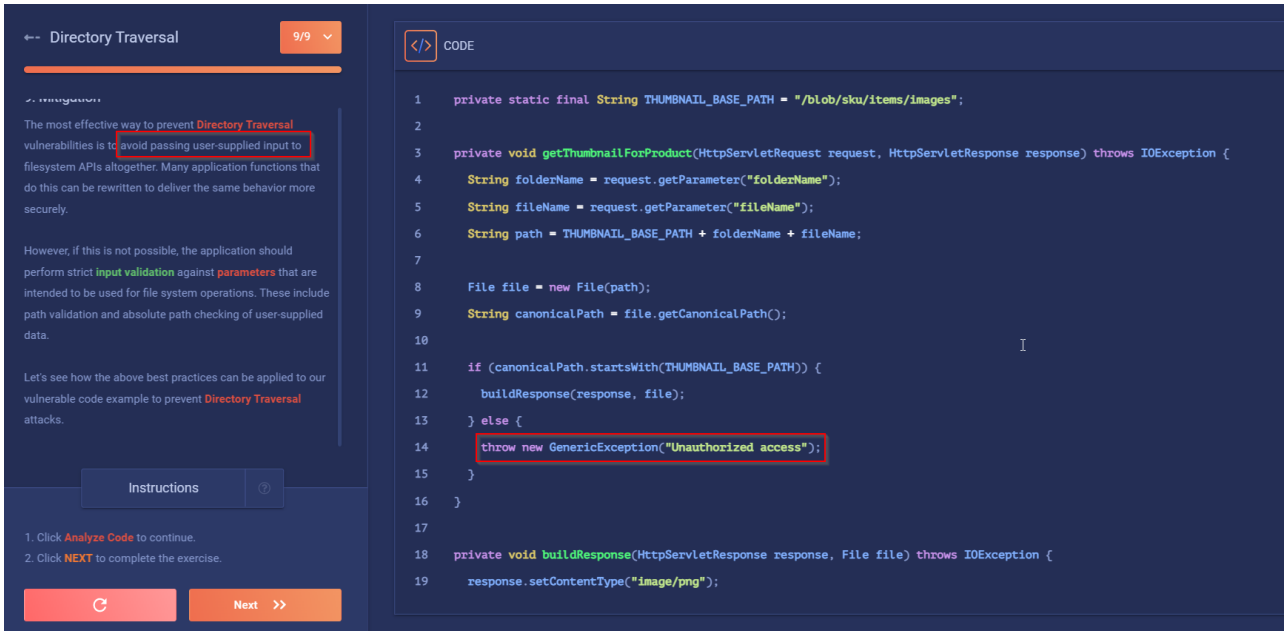


So, I tried a traditional method/payload to confirm the presence of directory traversal and it executed successfully. We can see the sensitive data.



As a part of mitigation, we should avoid parsing such user-supplied input.

Instant exception should be implemented in such a way that if the user-supplied path is not as the base_path should throw an error using error handling.



==========

Portswigger

==========

<u>**Definition by Portswigger**</u>:

Directory traversal (also known as file path 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.

Reading arbitrary files via directory traversal

Hint: The image files themselves are stored on disk in the location **/var/www/images/**

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

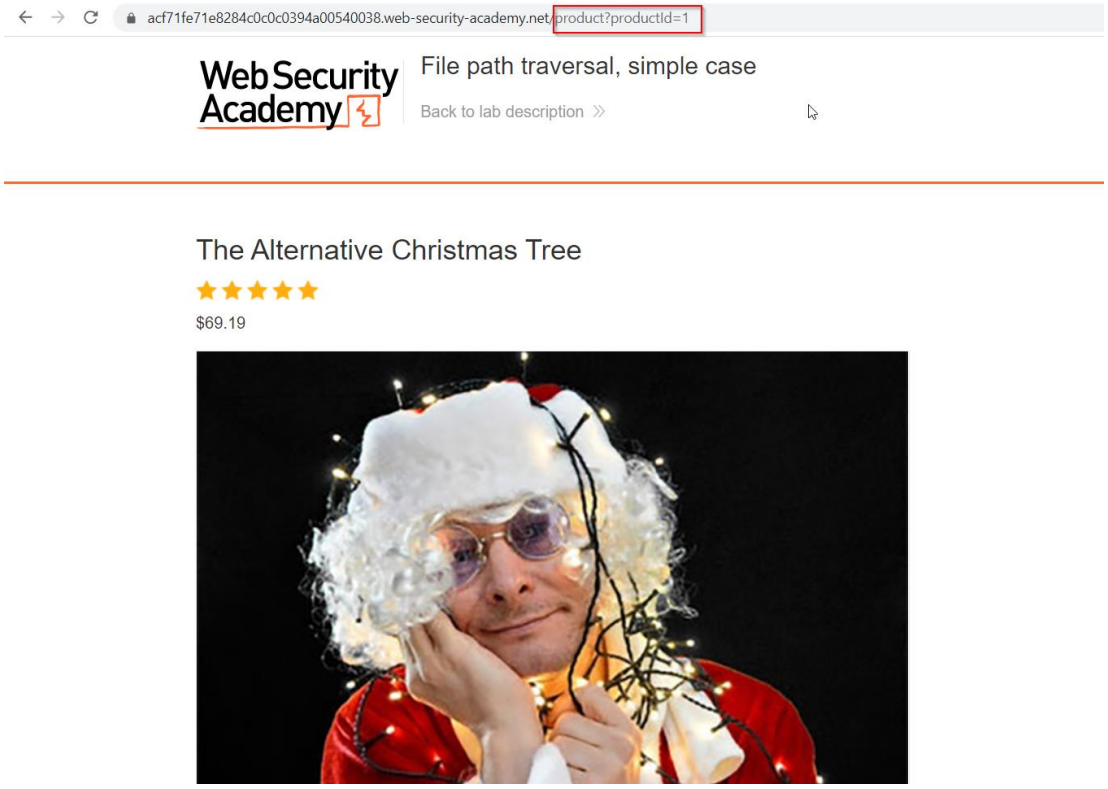
Example for windows system:

https://insecure-website.com/loadImage?filename=..\..\..\windows\win.ini

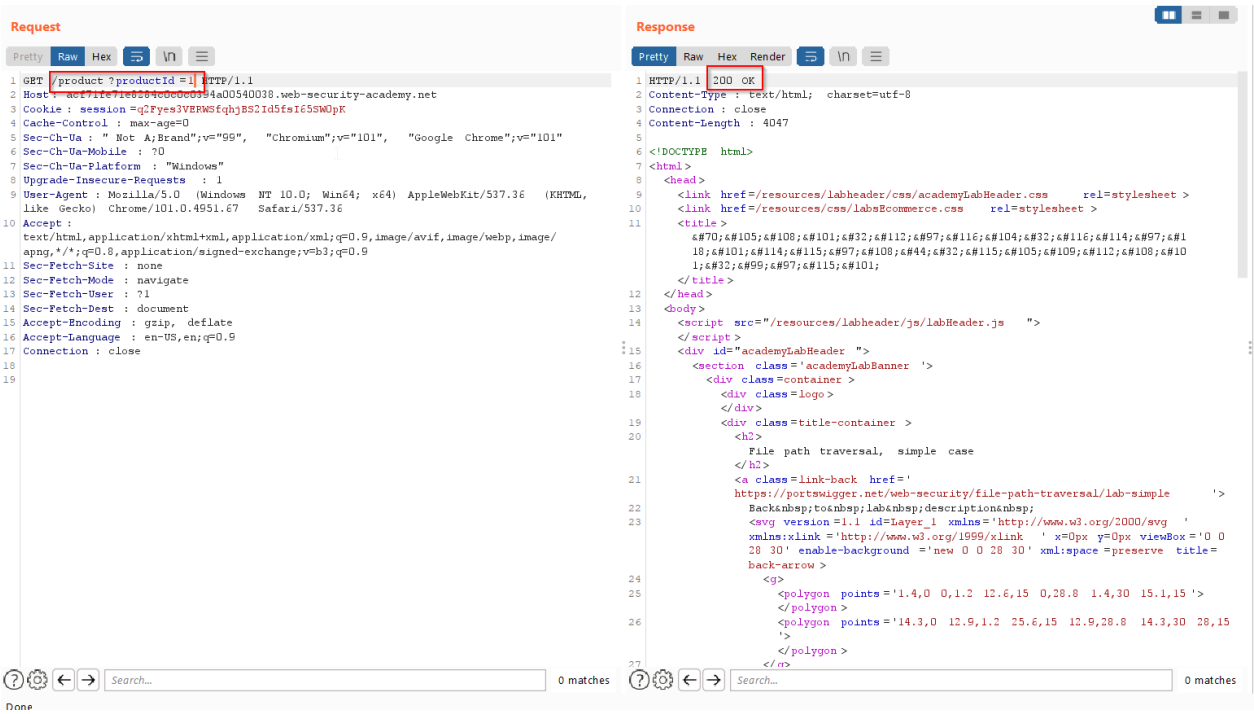
LAB 1: File PATH TRAVERSAL

Same has explained above. I found one functionality in website. which was used to retrieve a product data when clicked on product picture.

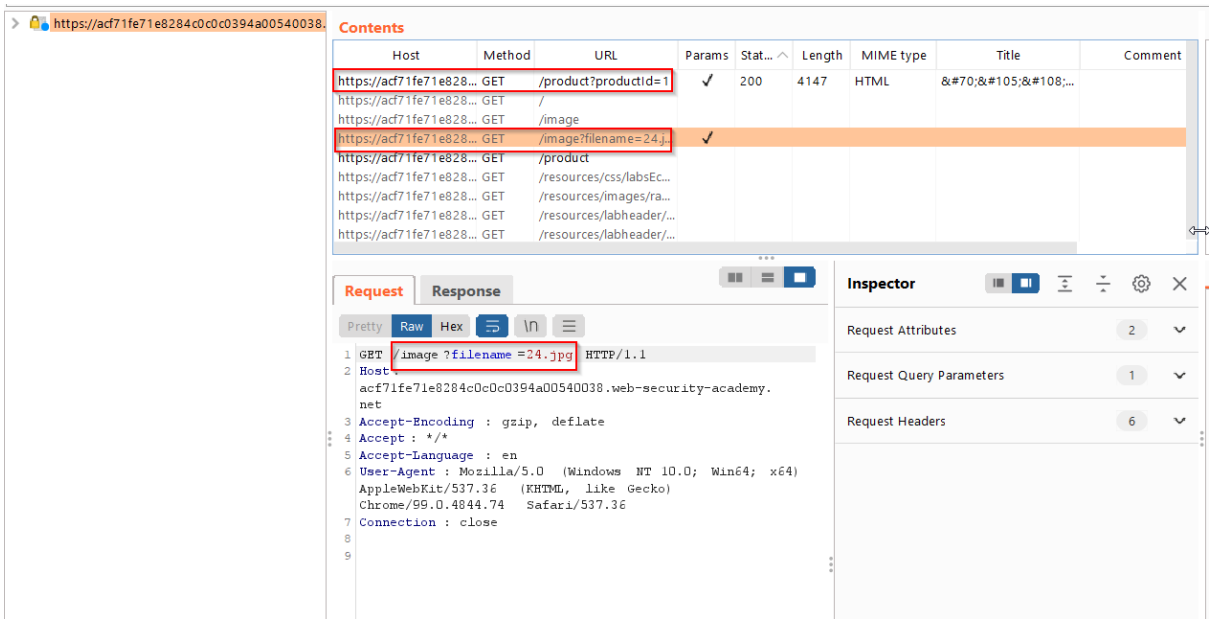
Where I observed that product was retrieved on basis of product id and In this case it was 1.



So, I have simply intercepted that request for testing but there was nothing to identify.

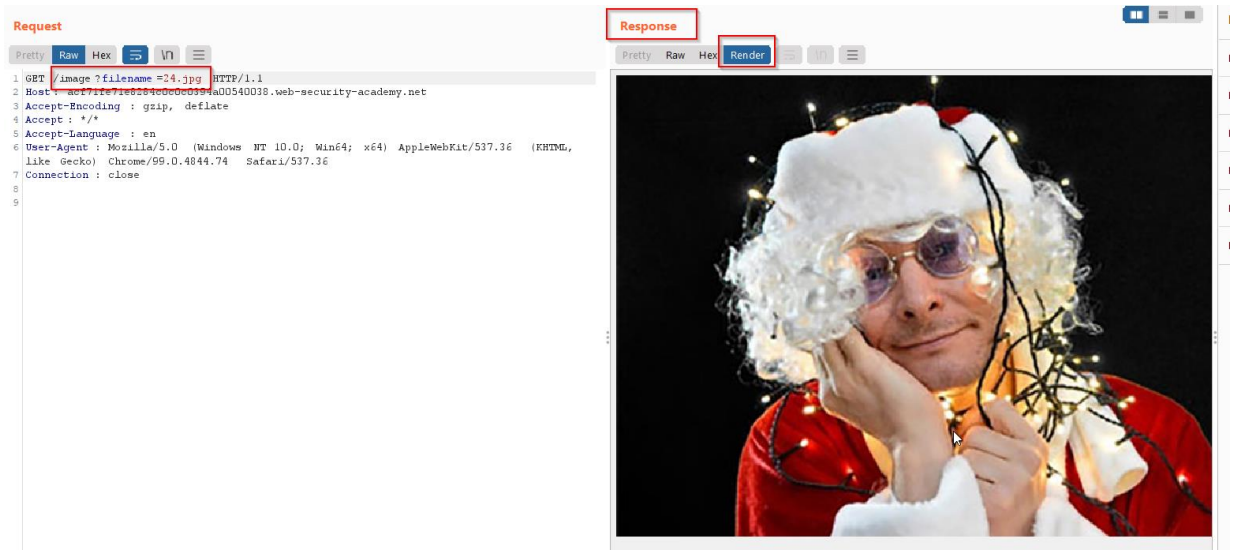


When checked in HTTP history. I found one entry which was pointing/resolving a url with filename parameter having .jpg has an extension.

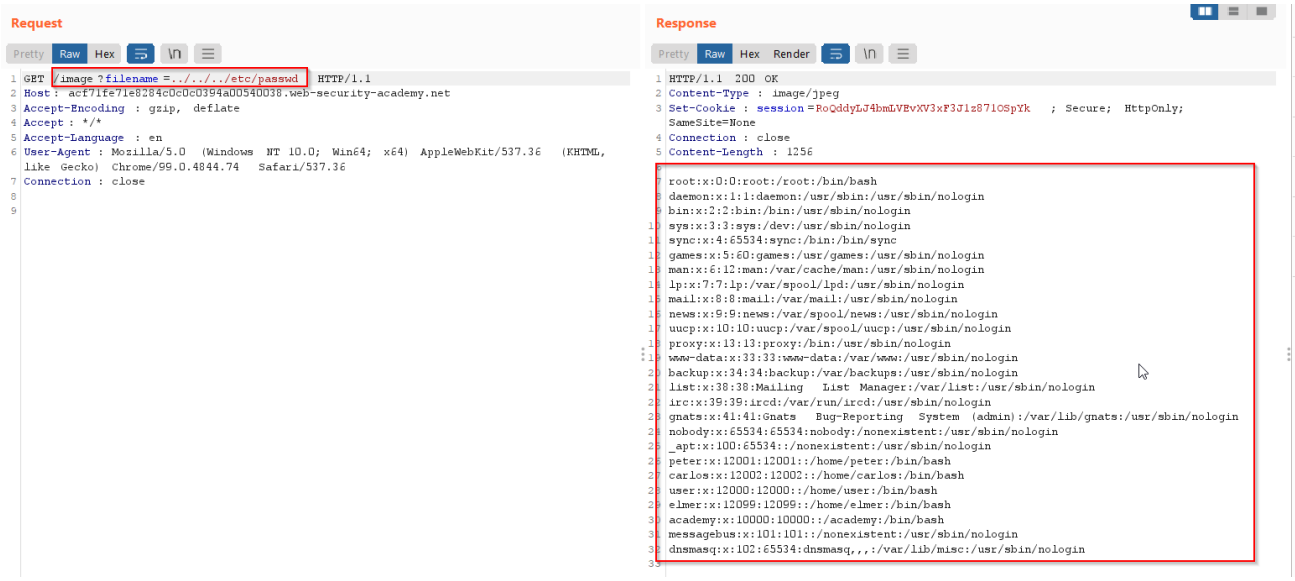


So, I tried testing on that by using send to repeater.

Let's modify the same and analyse before that lets have a look at response.



As it was working fine. I tried default payload for directory traversal which will lead to sensitive data exposure and guess what we found that in the response body. Refer below artifact for same.

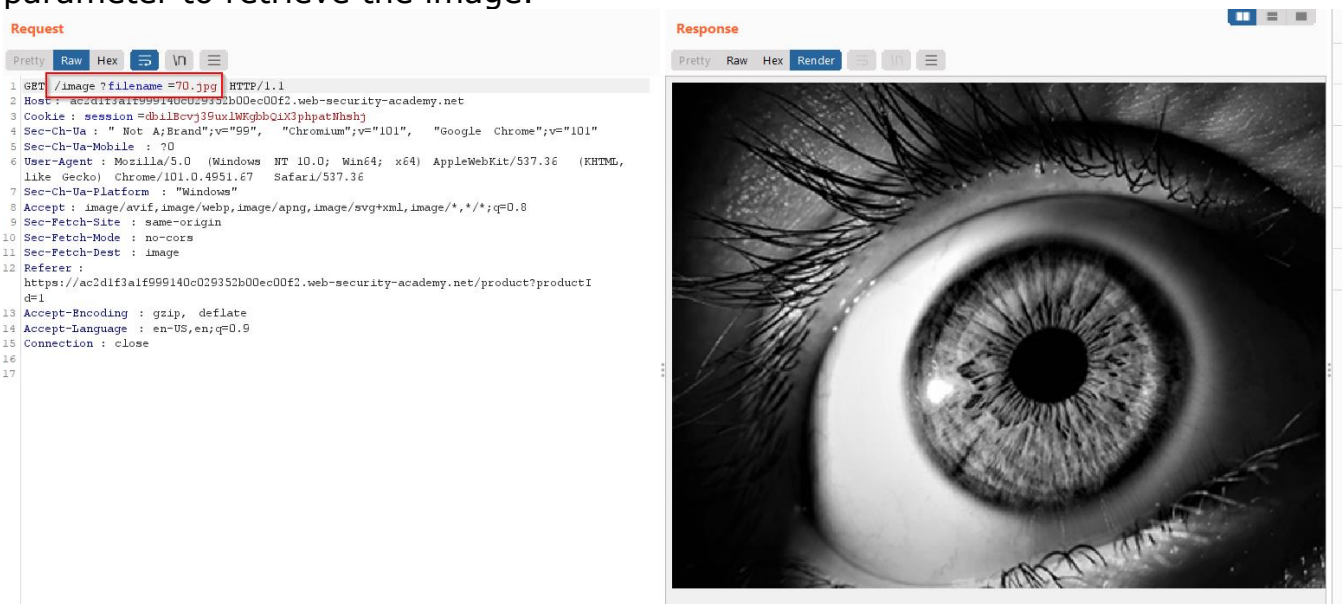


==========

LAB 2: Common obstacles to exploiting file path traversal

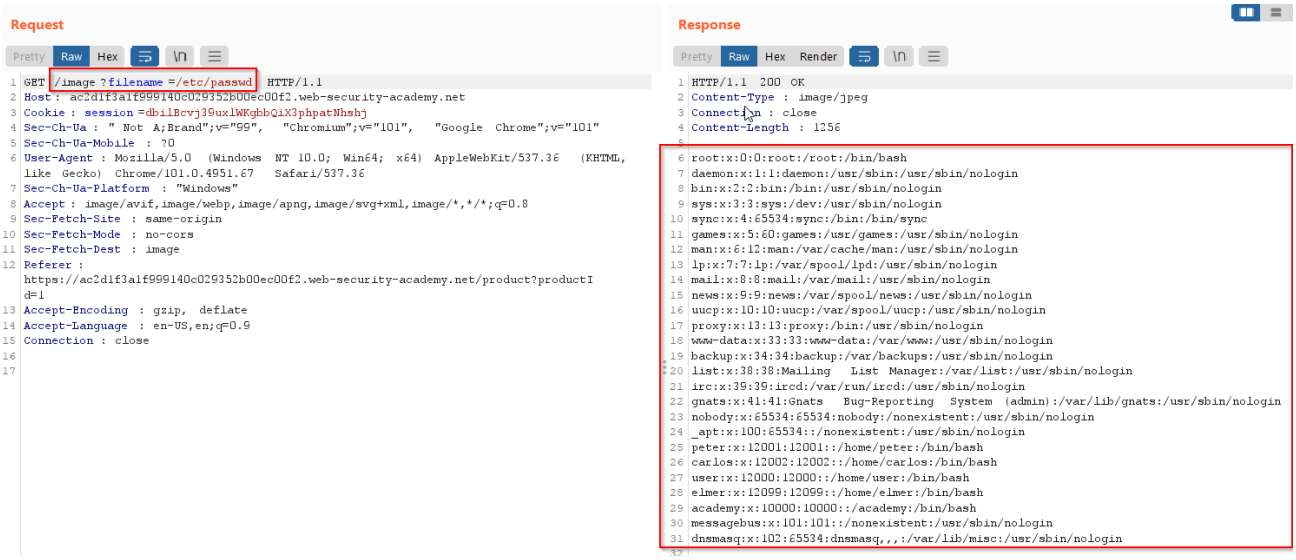
==========

Same has above in this lab we found a URL pointing towards a filename parameter to retrieve the image.



When forwarded the request in intercept. I found this filename request and I have sent it to repeater.

But now this time I have directly used /etc/passwd and it worked.



This was not that much interesting but here it was just to showcase that the direct path for retrieving the data can also be supplied if it is in same directory.

==========

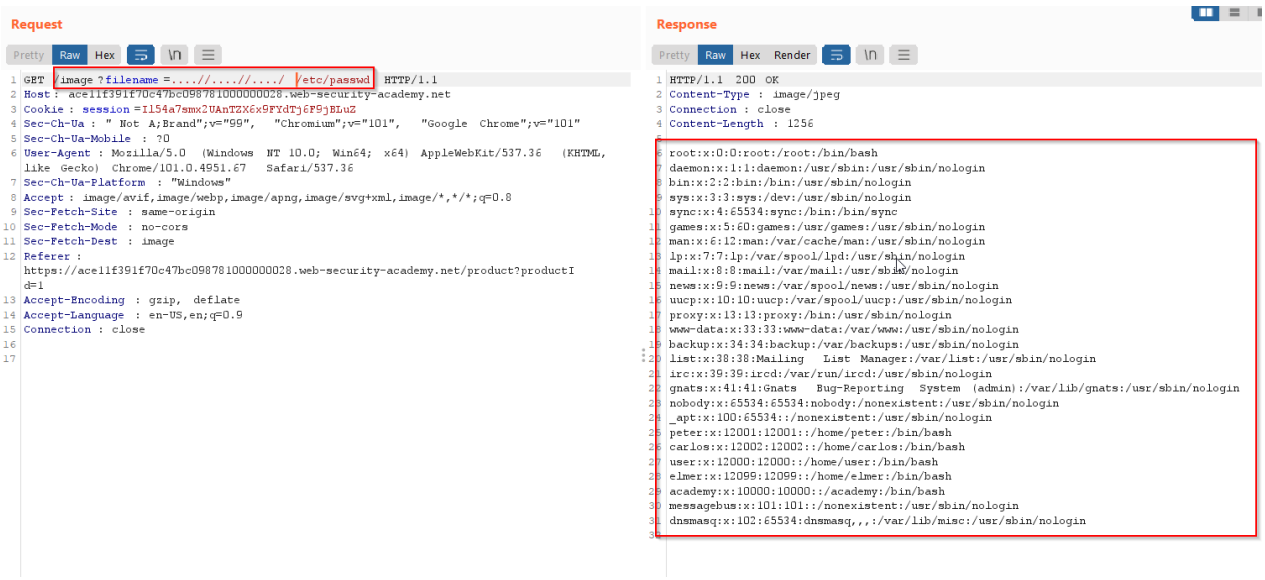
File path traversal, traversal sequences stripped non-recursively

==========

Hint: nested traversal sequences, such as ....// or ....\/

In this LAB, When tried with traditional payload we found that it was detected and forbidden the request. which in turn confirms the blocking of suppled ../../ traversal sequence. So we used ....// which when filtered out with ../ will return ../ which is required.

When tried, I was not facing any file error and it worked.



==========

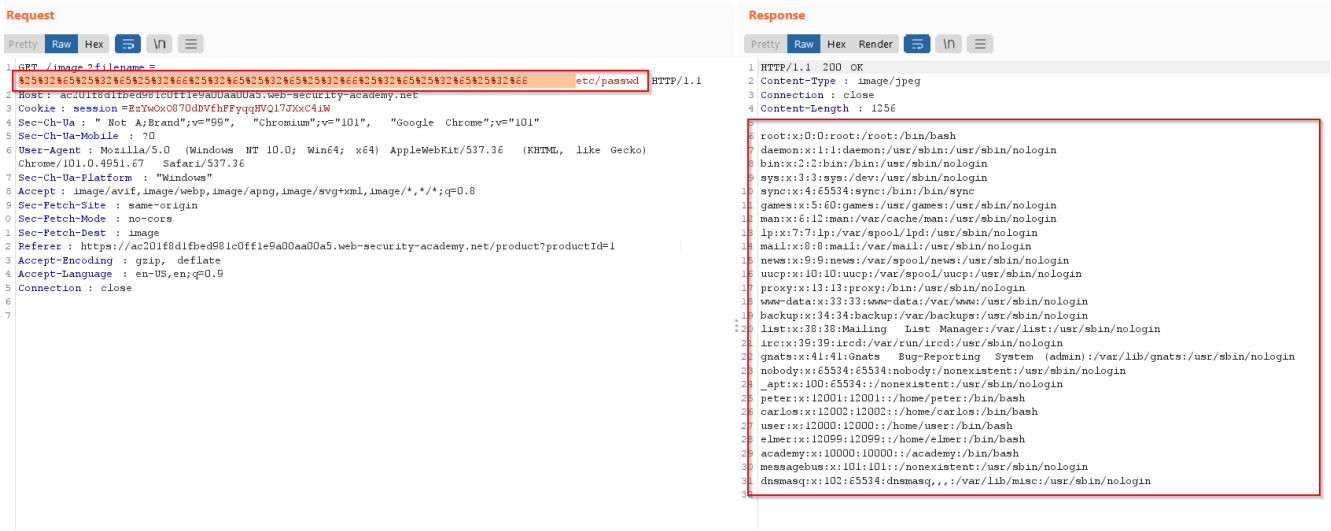
File path traversal, traversal sequences stripped with superfluous URL-decode

==========

***Hint*:** double URL encoding, The ../ characters when encoded and double encoded resulting in %2e%2e%2f or %252e%252e%252f respectively. Various non-standard encodings, such as ..%c0%af or ..%ef%bc%8f, may also do the trick.

When tried all of the above sequence we found that it was detecting and troughing an error.

So, I used double encoded on given traversal sequence ../../../ which was un detectable and we found our data.



==========

File path traversal, validation of start of path3

==========

This time we found that full path was used to retrieve the image file.

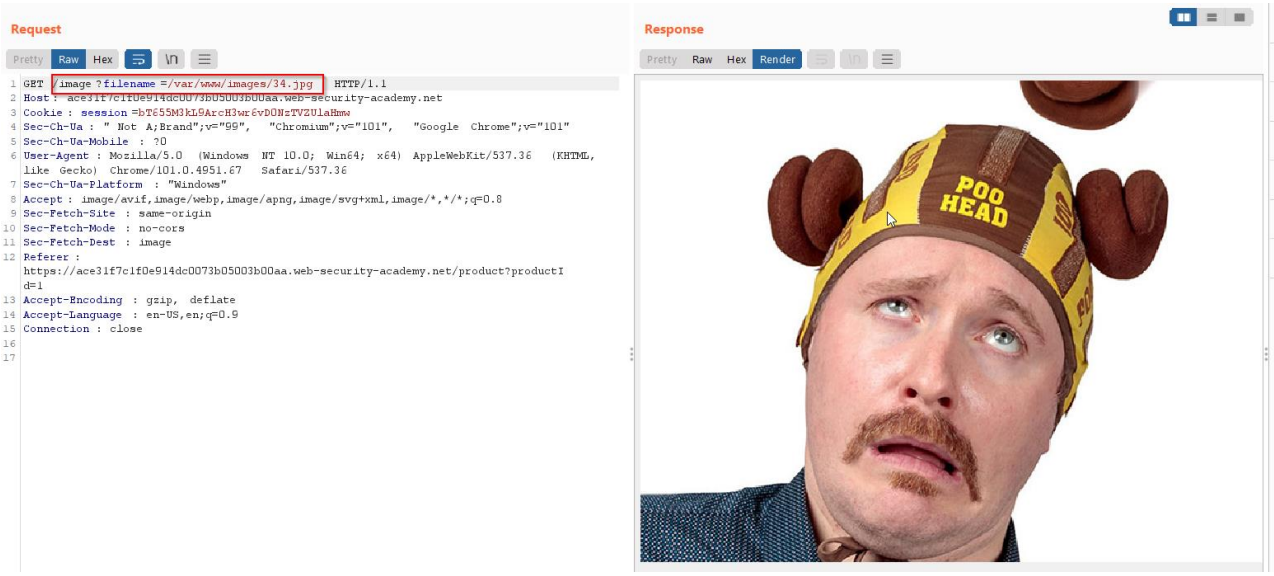
Which helps to predict and conclude the no of traversal sequence to use.

Given path: /var/www/images/34.jpg

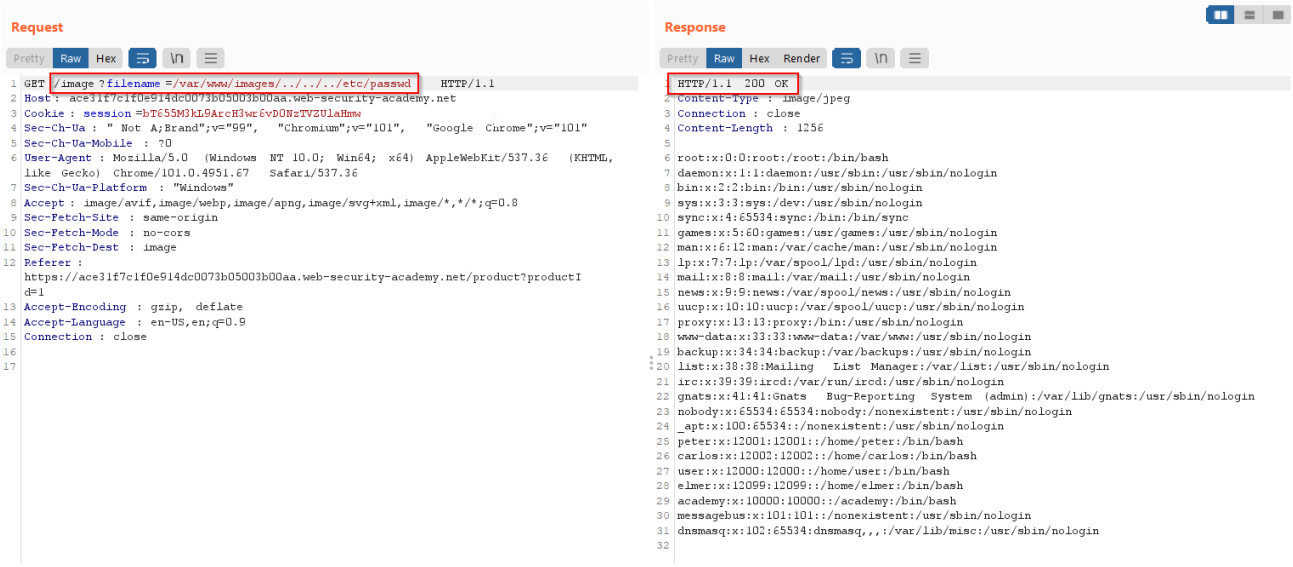
Required path: /etc/passwd

For required path we have to first move back 3 directories to come on / directory and from where will search /etc/passwd.

Normal request containing path



Adding required path in request.



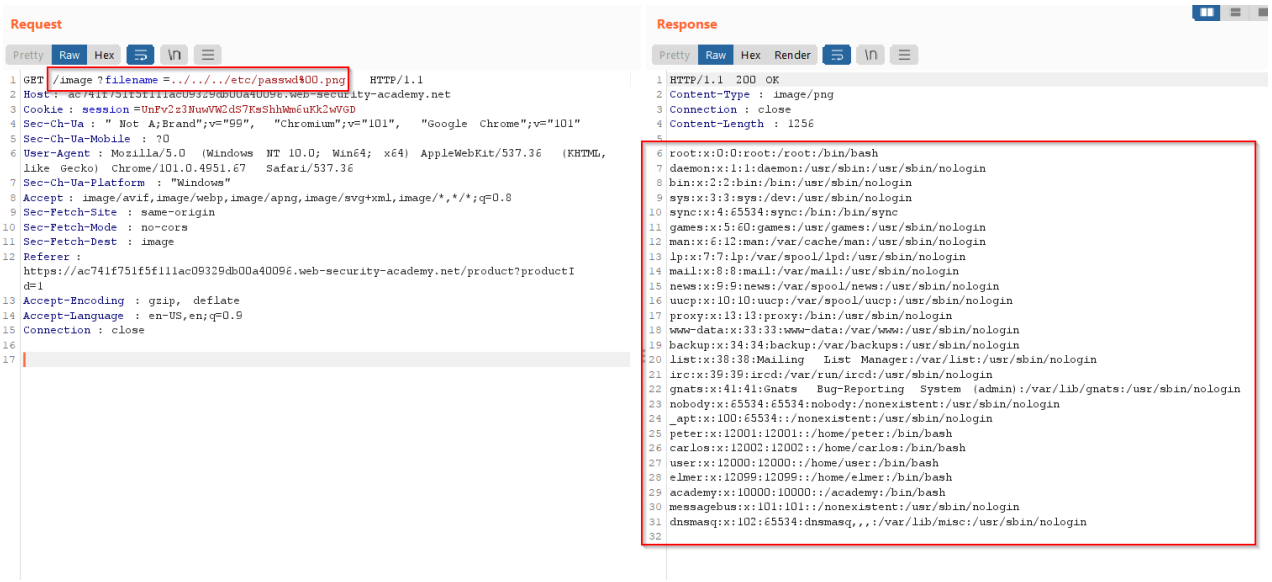
==========

File path traversal, validation of file extension with null byte bypass

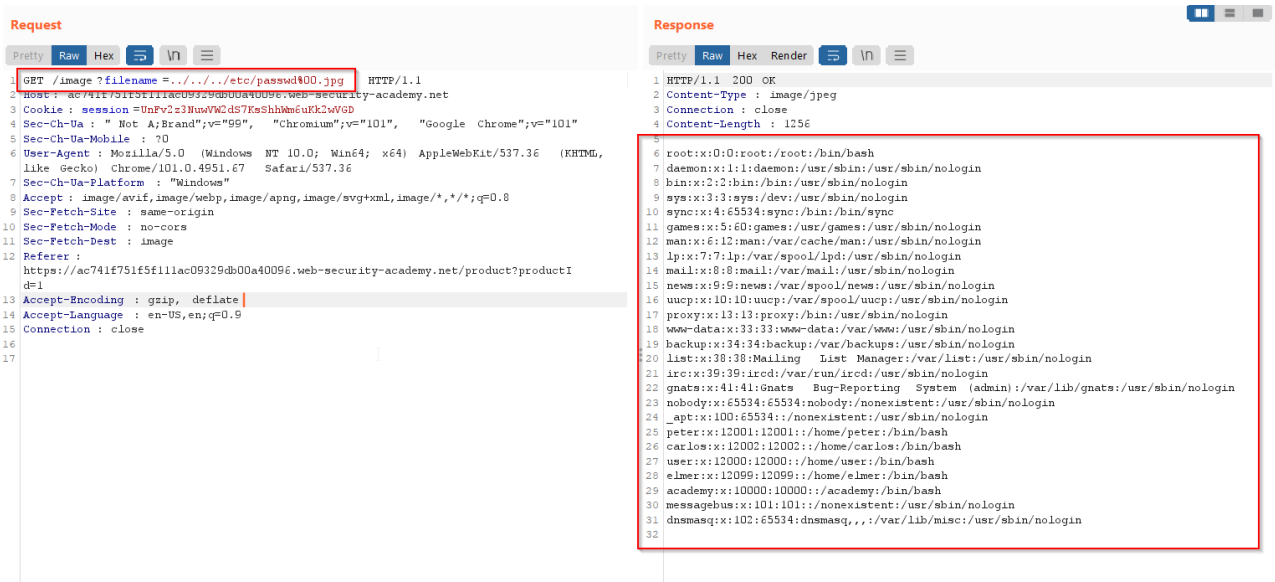
==========

<u>Null byte</u>: %00

Here it was validating the request with the file extension. So, when tried with normal /etc/passwd it was not working so we appended a null byte with required file extension that is .png and it worked.



Just to come the scenario I tried .jpg extension and it also worked.



Prevention:

a. avoid passing user-supplied input to filesystem APIs
b. The application should validate the user input before processing it
c. It should verify that the canonicalized path starts with the expected base directory.

THE END