iGNITE Technologies

# DETAILED GUIDE ON


# PATH TRAVERSAL

www.hackingarticles.in

# Contents

# Introduction

In our previous post, we've explained the **Local File Inclusion attack** in detail, which you can read from **here.** I recommend, then, revisiting our previous article for better understanding, before going deeper with the **path traversal vulnerability** implemented in this section.

Today, in this article we will explore one of the most critical vulnerabilities that arises when the developer does not validate the inclusion functions in the web-applications, which thus allows the attacker to read and access any sensitive file from the server.

# Introduction to Path Traversal

**Path Traversal** sometimes also termed as **"Directory Traversal"** is an HTTP vulnerability which allows an attacker to trick and manipulate the web application's URL to access the files or directories that resides outside the application's root folder. This vulnerability carries when a developer fails to establish or manage the input validations while including files such as images, static texts, codes, etc. in their web applications.

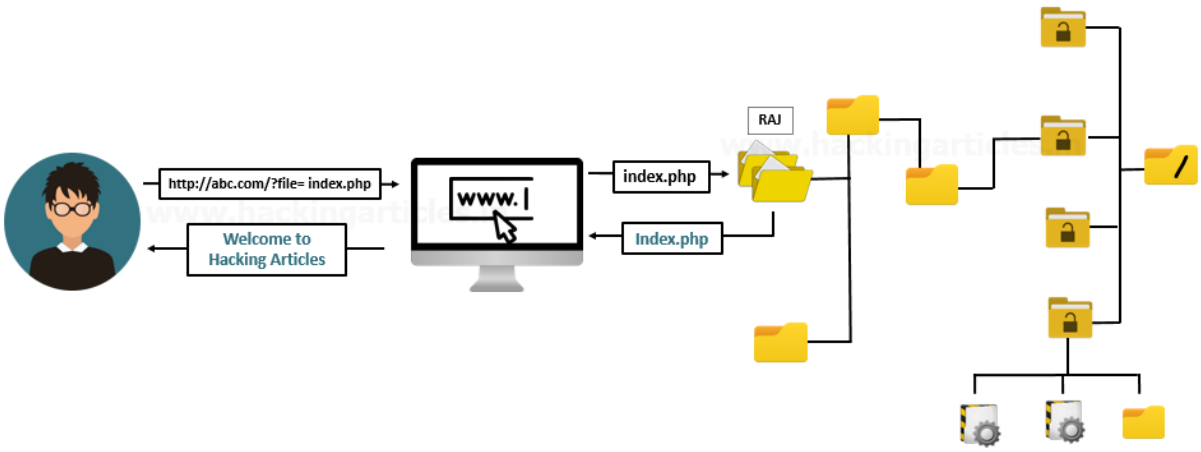
However, in such attacks, the attacker manipulates the web application input fields by entering **the dot-dot-slash (../)** sequences or some similar variations, to bypass the web page and access the desired system file.

Thus, this vulnerability has been reported as **"High** with a **CVSS score of 7.3"** under:

1. **CWE-22:** "Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')"
2. **CWE-35:** "Path Traversal: '.../...//' "
3. **CWE 73:** "Directory Traversal"
4. **CWE-200:** "Exposure of Sensitive Information to an Unauthorized Actor"

Let's check out this scenario and learn how an attacker defaces the web-application by grabbing the server's sensitive files.

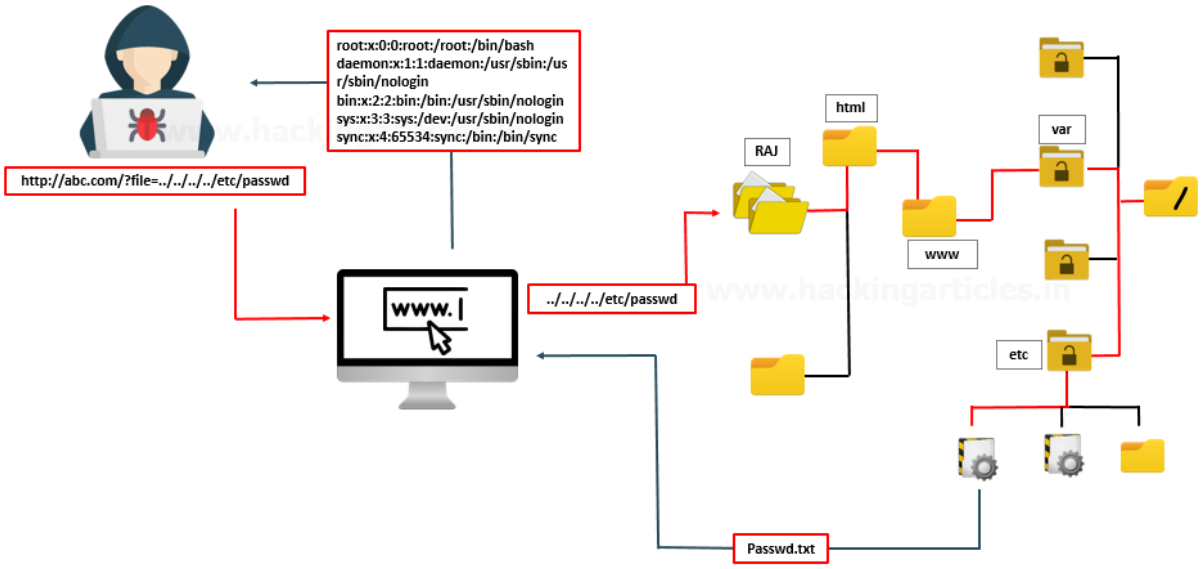
*Here, the user calls up a file - **index.php** through the web application's URL i.e. **http://abc.com/file=index.php.** Thus, the application processes the URL and calls up the **index.php** that was present locally into the server folder **"RAJ"** as "**/var/www/html/RAJ".***




*The developer uses the "include" functionality as **"file="** with a simple intention to manage the user's selected input files, such that the application can directly call it from the local server. Now the*

*attacker tries to manipulate the URL using the dot-dot-slash sequence as* ***http://abc.com/file=../../../../etc/passwd,*** *to retrieve the contents of the server's password file.*

*Thus, again the application will process it and read up the file at* ***/var/www/html/RAJ/../../../../etc/passwd.*** *Every* ***"../"*** *represents - back to parent directory, thus if we call up "../" for four times, it will put us in the* ***"root"*** *directory, from there we can simply access the password file as* ***etc/passwd****.*




## Linux Server Path_Traversal Exploitation

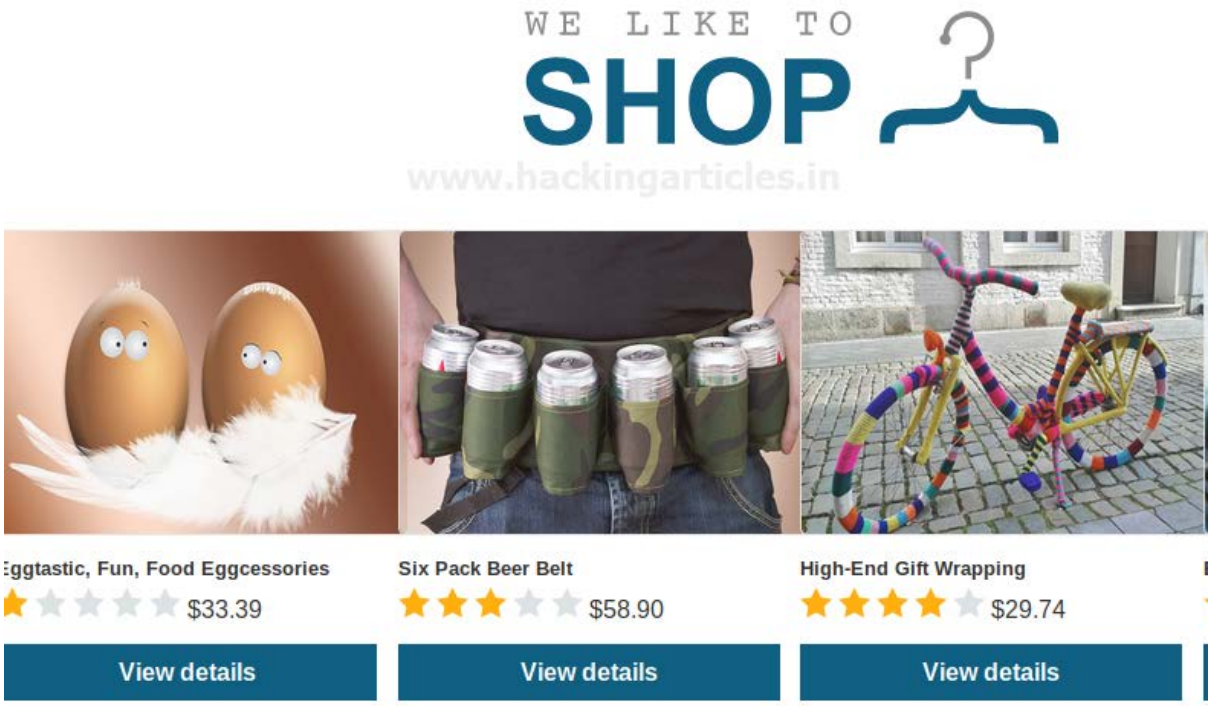
Let's now try to implement this in some real scenarios and check the different attacking sequences rather than the dot-dot-slash only.

For all this, I'll be using two different platforms [The Portswigger Academy] and **DVWA** which contains the path traversal vulnerability.

### Basic Path Traversal

Login into the **PortSwigger academy** and drop down till **Directory Traversal** to get into its labs, choose the first lab as "**File path traversal, the simple case"** and hit the **"Access the lab"** button.

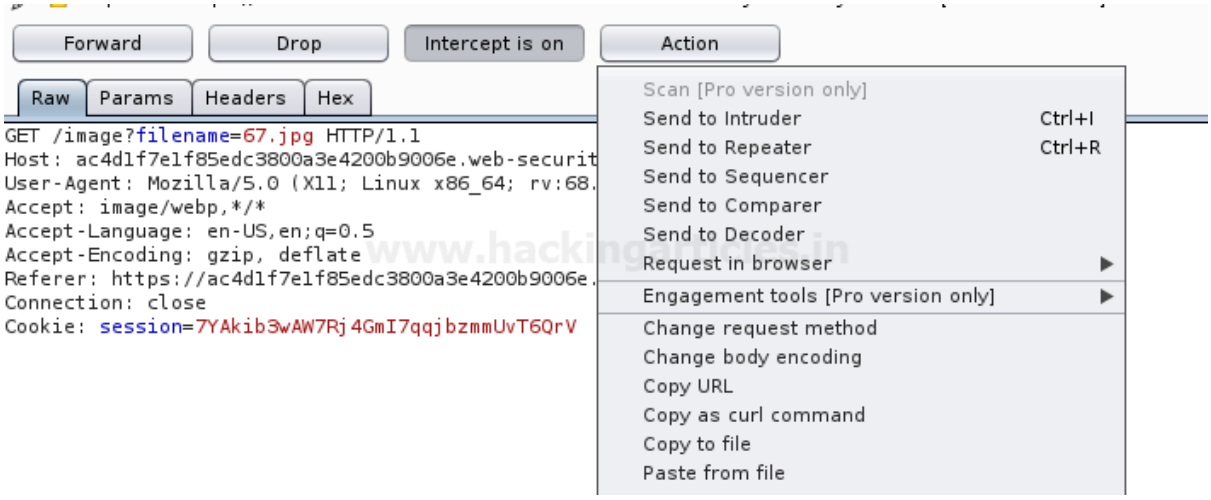
Here you'll now be redirected to an e-commerce website, which has several products in its catalogue and is suffering from path traversal vulnerability.




As to further, I've opened a product and checked out its display image with a simple right-click as **view image.**

Now it's time to check what we could manipulate.

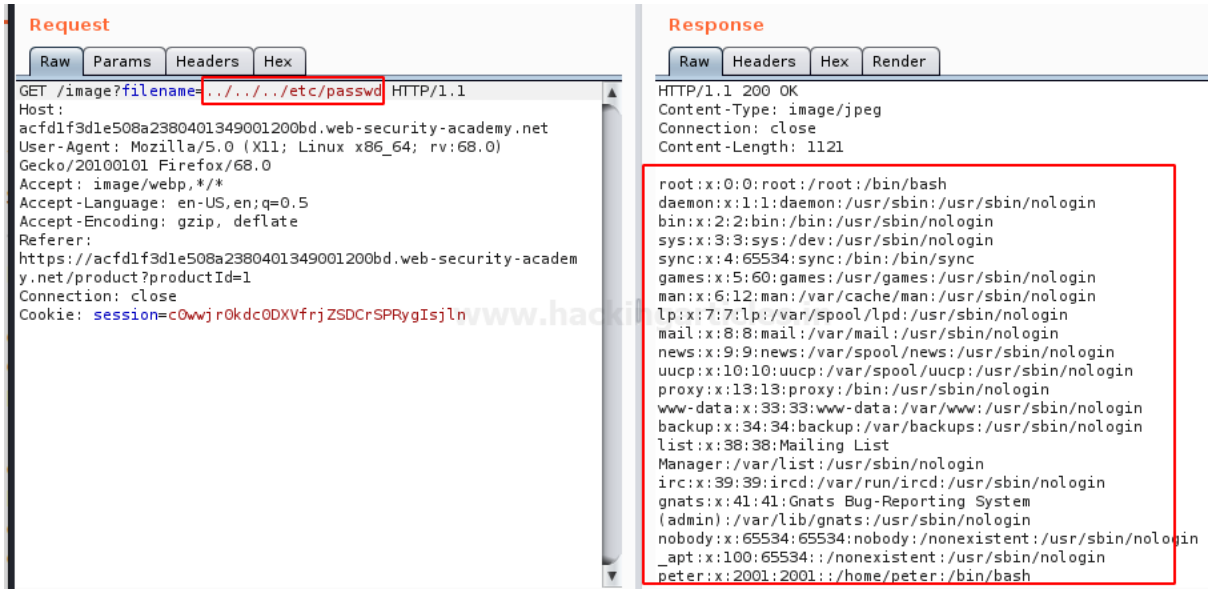
Tune in your Burp Suite to capture the ongoing **HTTP Request** and share it all with the **Repeater**.




As in the GET request, above in the image, you can notice that the **filename=67.jpg**, let's try to change this filename with

```
filename=../../../etc/passwd
```

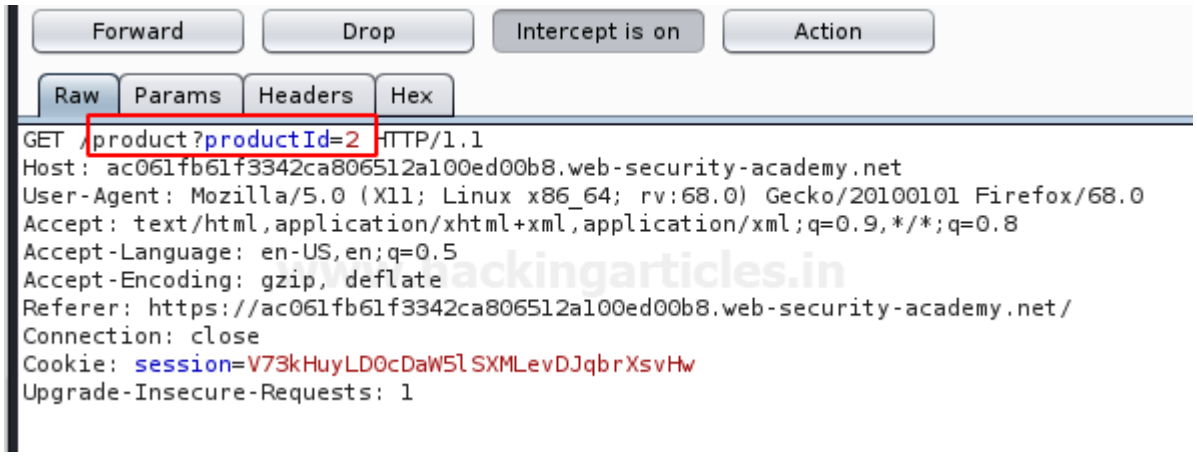
Great!! From the image below, you can see that we've successfully grabbed the **passwd** file.



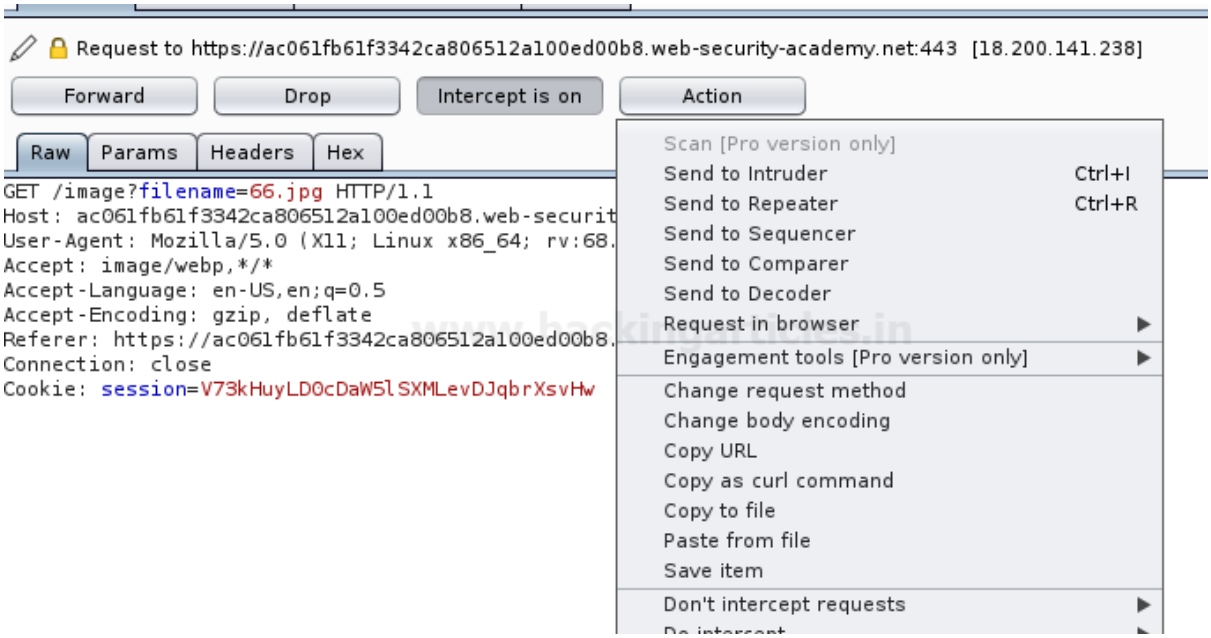

## Blocked Traversal Sequence

There are situations when the developers end the traversal process, i.e. the dot-dot-slash or any subsequent sequence will not work in such case.

While getting to the second lab, I got the same issue i.e. the "**../**" sequence didn't work, and I fail to capture the password file. So, let's try to capture this request again in our burpsuite monitor.

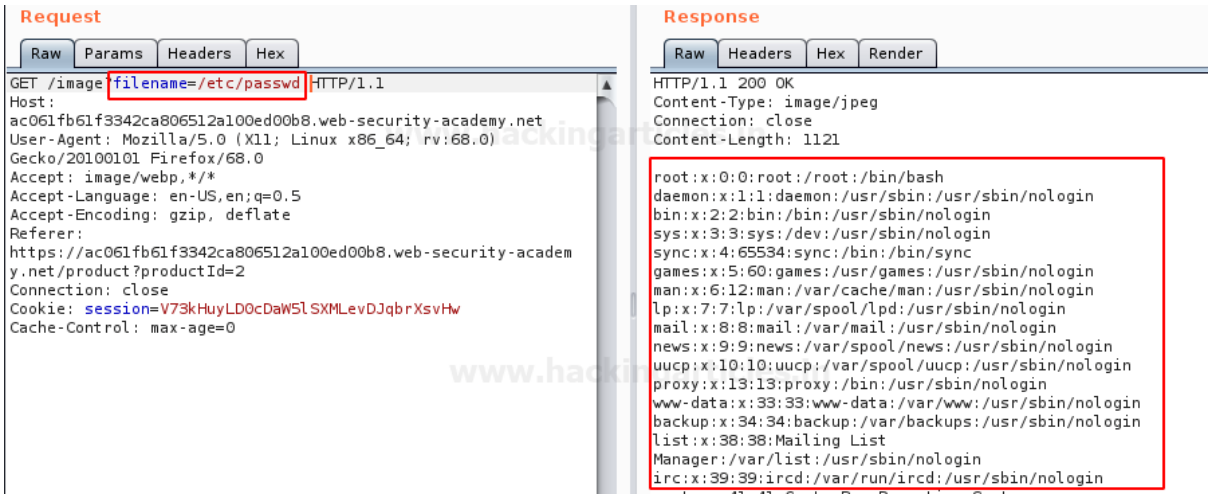



From the image below, you can see that I've grabbed up the request with **filename=66.jpg,** and now I will shift this all to the **Repeater**.




As we're blocked with the "../" sequence. Let's try to enter **/etc/passwd** without any preceding values.

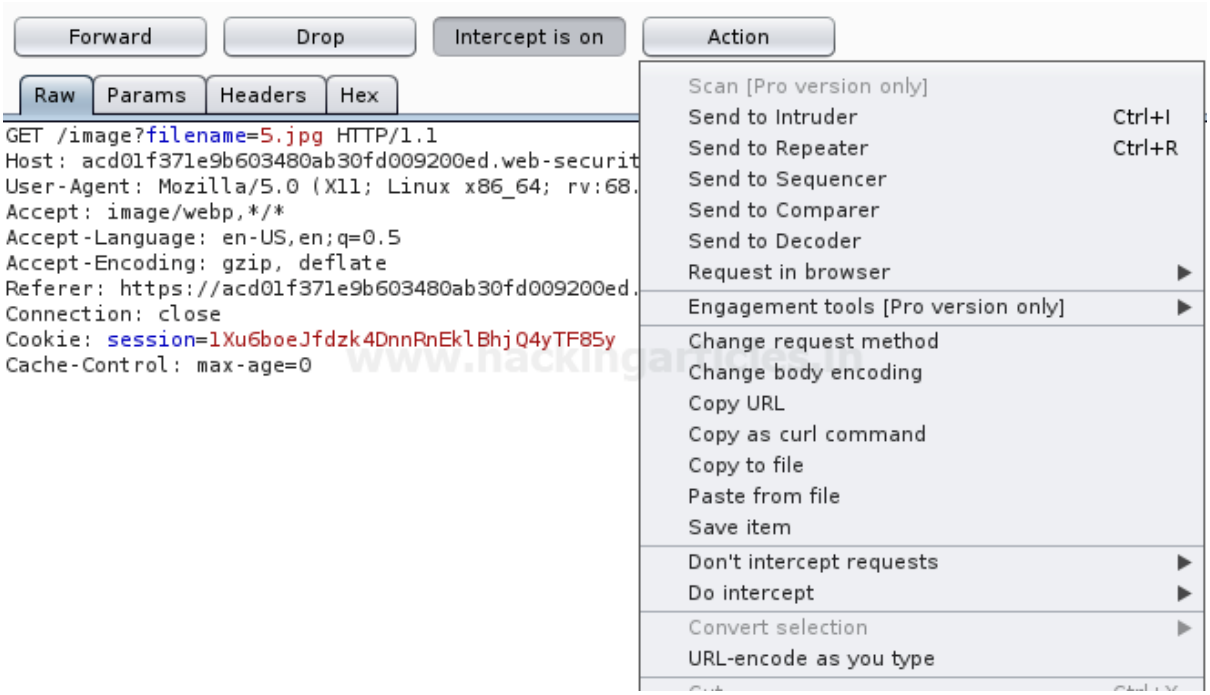
Cool!! This worked, we got the **password** file with the direct call.




## Validated Path Traversal

Many developers validate their web-applications, that if the **"../"** comes into the URL, it gets rejected out. Thus, when we tried both the above procedures in our next lab, we got rejected and didn't grab anything.

Therefore, we capture the **HTTP request** in our Burpsuite and traverse it to the **Repeater**.




This time we manipulate the URL filename parameter with **"double dots followed by double slashes"** i.e. "**....//....//....//etc/passwd**"


**Great!!** From the above image, you can see that we've again captured the password file with this unusual technique.

As we jumped over the **4th lab,** we got this, the developers had made a validation which blocks up the input which contains the path traversal sequence.

# Lab: File path traversal, traversal sequences stripped with superfluous URL-decode

PRACTITIONER

LAB Not solved

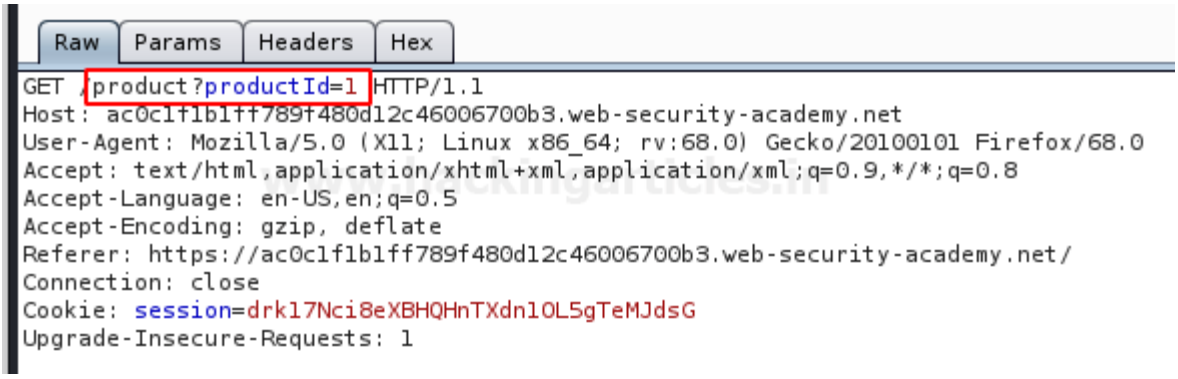
This lab contains a file path traversal vulnerability in the display of product images.

The application blocks input containing path traversal sequences. It then performs a URL-decode of the input before using it.

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

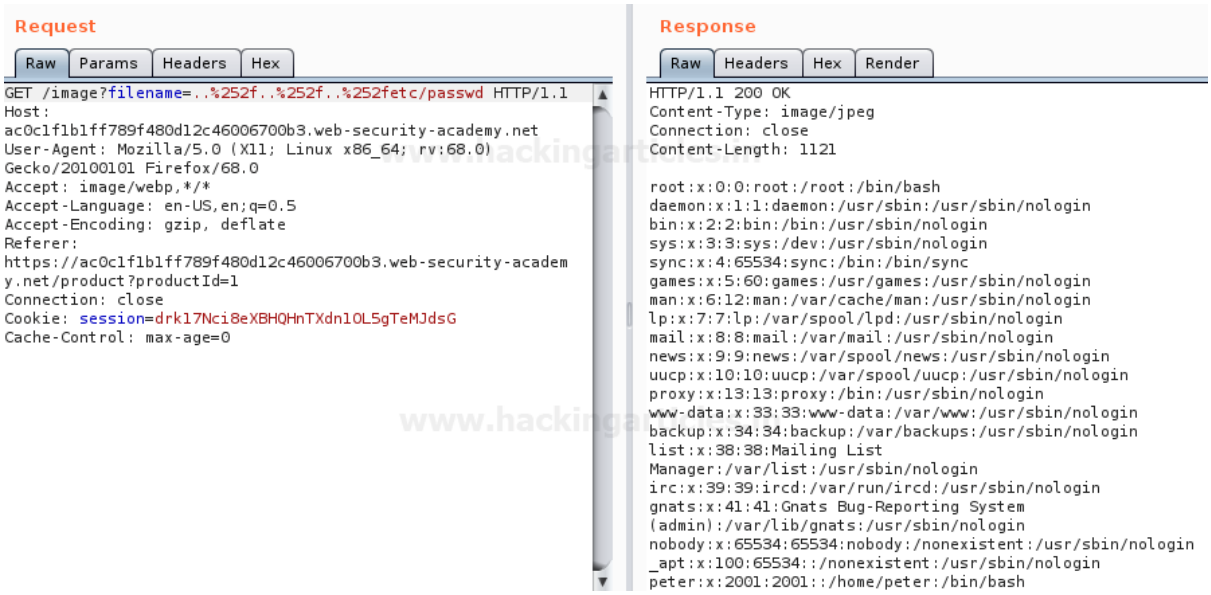
Access the lab

Therefore, to bypass this validation, I've again captured the request and send it to the repeater, to make some manipulations.




From the below image, you can see that, I've manipulated the URL filename parameter using Double URL encoding method in order to convert "../" into "..%252f" with the help of ASCII to URL encoder converter and have successfully accessed the password file with

```
filename=..%252f..%252f..%252fetc/passwd
```




## Path Disclosure in URL

*Isn't it great if you get the number of back steps you need to perform to capture your desired file?*

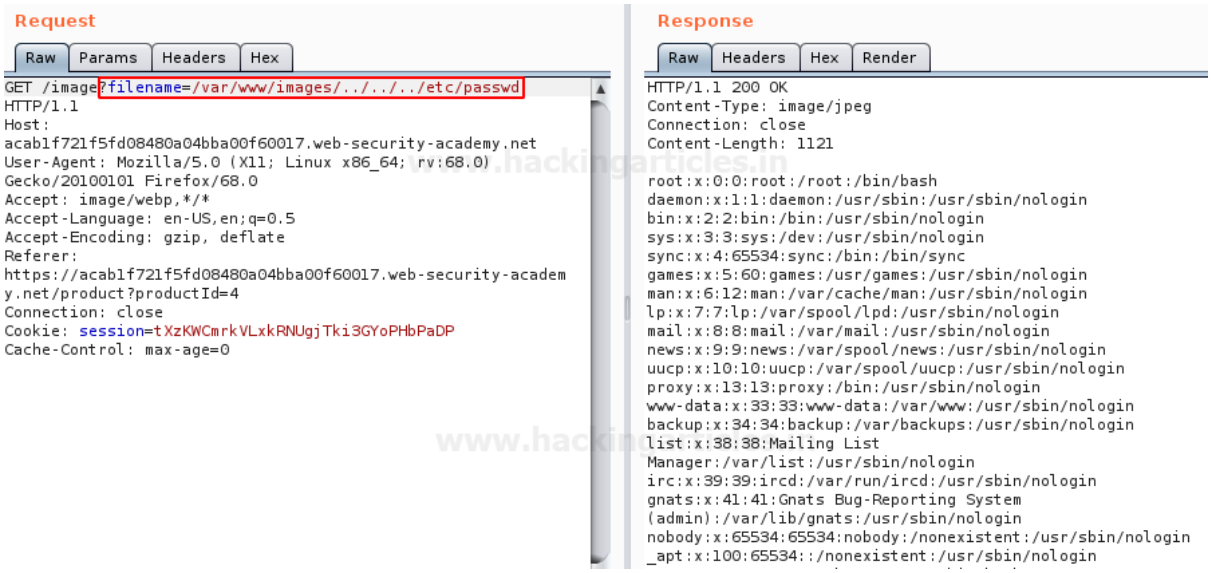
Path disclosure is that vulnerability, where the URL offers the complete path of the file it is containing, which thus allows the attacker to simply manipulate the URL and with no efforts he can access the system files.

As we moved further to lab **5,** we were encountered with an application that was offering us the **complete path of the file**.

We simply captured that request and sent it to the **repeater**. From the below image, you can see that the filename parameter is having the value as **"/var/www/images/21.jpg".** Which means that the file "21.jpg" is inside the **images** directory and the **root directory** is just **3 steps** away from us.


So, we are now aware of the number of back steps we need to make to get into the password file, therefore we'll do that as

| filename-/var/www/images/../../../etc/passwd |
|---|




## Null Byte Bypass

Many developers add up a **'.php'** extension into their codes at the end of the required variable before it gets included.

Therefore, the webserver interprets the **/etc/passwd** as **/etc/passwd.php**, thus we could not access the file. To get rid of this ".php" we try to terminate the variable using the **null byte character (%00),** that will force the php server to ignore everything after that, as soon as it is interpreted.

As soon as we share the captured request to the repeater we'll try to eliminate this null byte character as discussed above.


So, from the image below, you can see that we've again captured the password file by adding up (%00) in the URL as:

```
filename=../../../etc/passwd%00.jpg
```


# Windows Server Path_Traversal Exploitation

It’s not necessary that every time we encounter an application which is running over a Linux server, thus there are chances that our luck doesn't work, and we got stuck with a window’s server. Let’s learn the different sequences and the method that can be used during such situations.

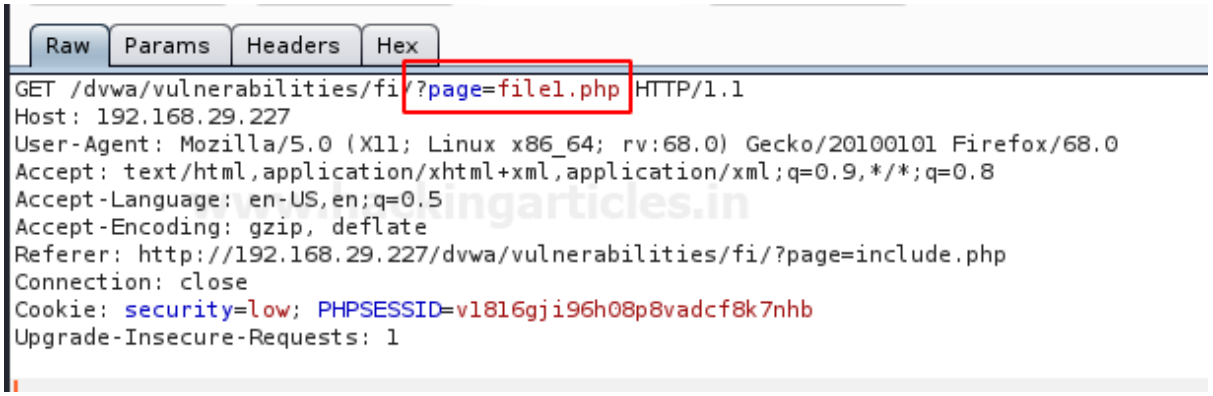
## Basic Path Traversal

I’m having **DVWA** setup over my **window’s machine**. You can learn this all from **here**.

Let's boot inside the DVWA application as **“admin: password”** with the security level as **“low”.** Further, choose the vulnerability as **File Inclusion** from the left-hand panel.

As soon as we choose this, we’ll be redirected to the webpage which is suffering from **path_traversal vulnerability.**

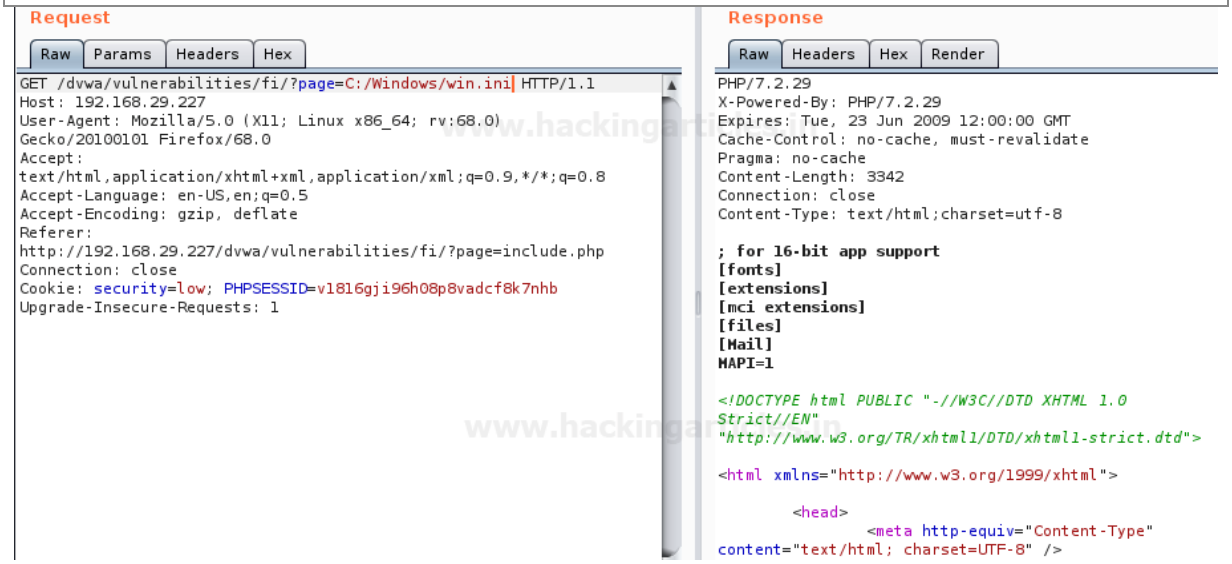
Let’s capture this request through burpsuite and see what we can get through it.



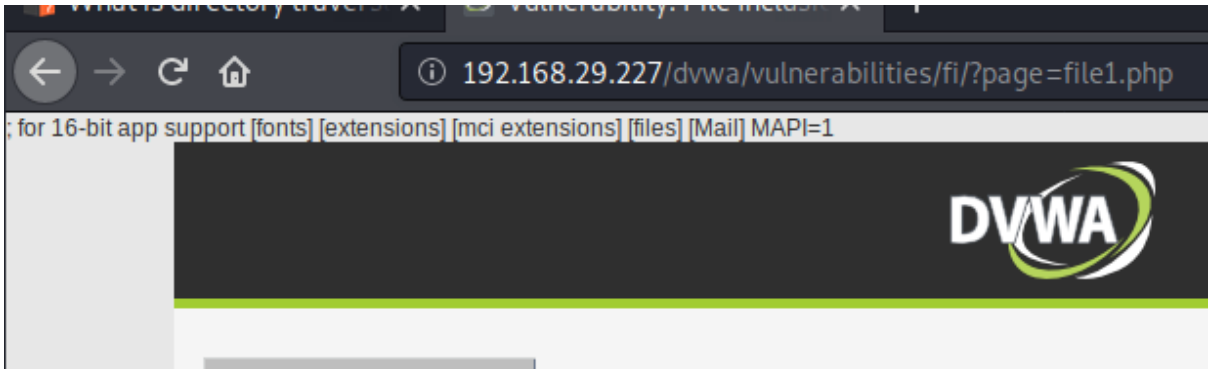

From the above image, you can see that **file.php** is included in the **page parameter.** Let’s share this all with the repeater and try to play with this input field.

To call up the **windows file** on the web-applications screen, manipulate the page parameter with the following input.

page=C:/Windows/win.ini



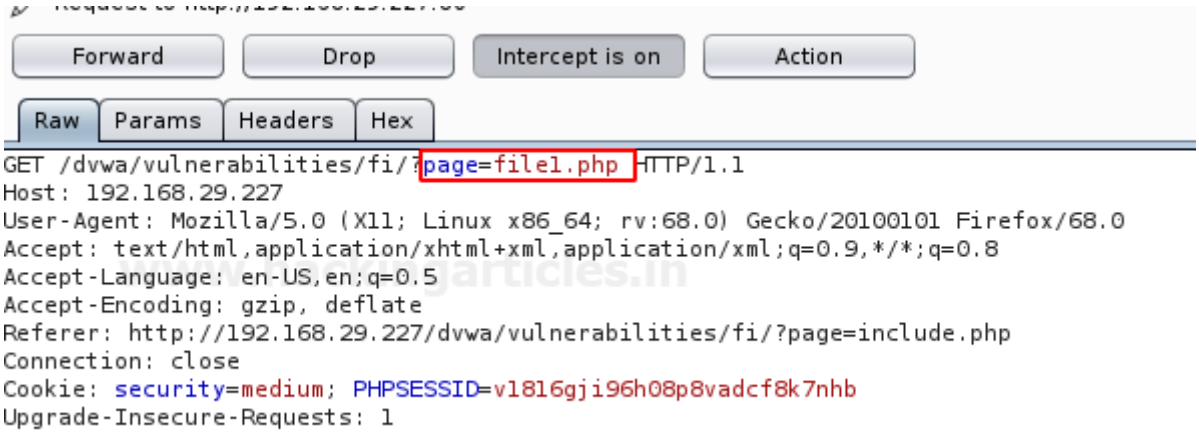

From the above image, you can see that we've successfully called up the file in the response tab. Now **forward** this request and check the result over the application's screen.




## Double dots with Forward-Backward Slashes

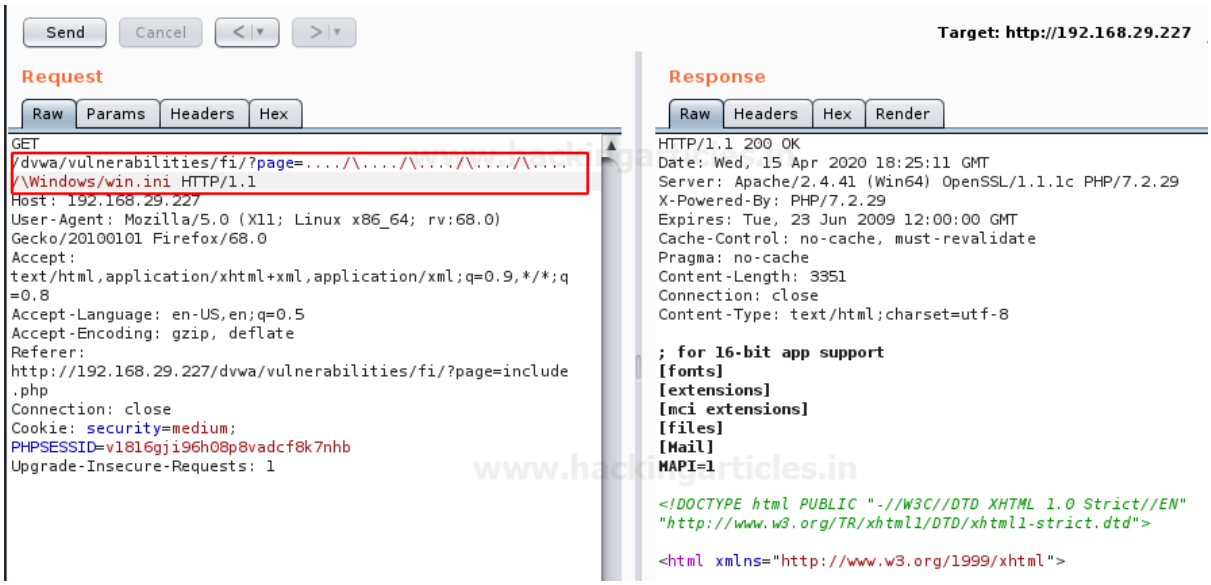
Whether the application is hosted over a Linux server or a window one, the developers always validate their web-applications. Here, to keep the application secure with the path traversal attacks. the developers block up some sequences such as **"../"**, which thus gets rejects out automatically if entered in the URL.

Increase up the DVWA's security level and set it too **"medium".** Capture the request at burpsuite and send everything directly to the repeater.



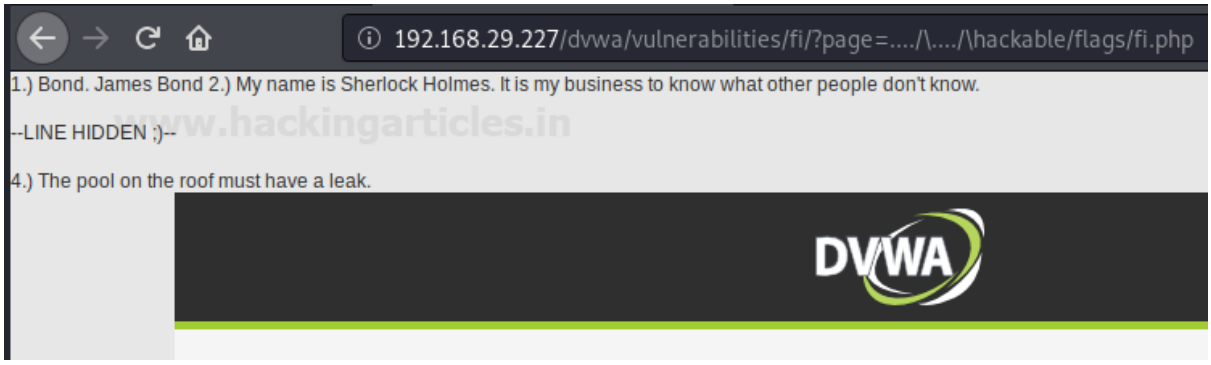

Form the below image, you can see that we've successfully bypassed this validation by **the double dots** followed by **forward-backwards slashes** and have again grabbed the **"win.ini**" file by :

```
page=..../\..../\..../\..../\..../\Windows/win.ini
```




Using a similar sequence, you can even capture other files present in the windows system. From the image below you can see that I've grabbed up a flag i.e. **fi.php** which resides in the hackable folder by simply manipulating up the URL parameter as:

```
page=..../\..../\hackable/flags/fi.php
```




## Blocked Traversal Sequences

There are many situations when such conditions didn't work, that is the developer validates and blocks every possible sequence he can.

Let's find the other possible way to get the **"win.ini"** file without getting involved with the commonly used sequences.

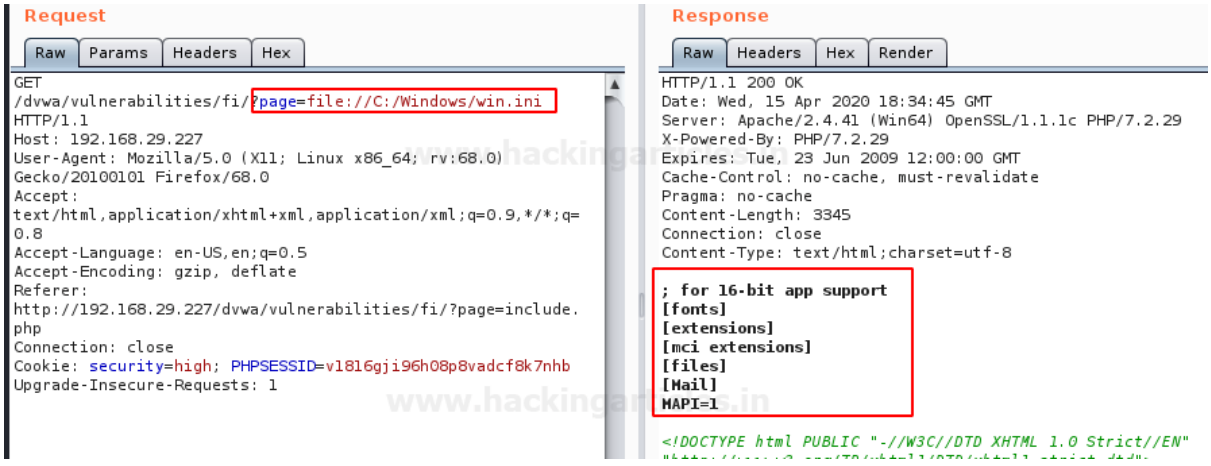
Again, go for the security option and hit it up with the **high security** in your **DVWA** application. Come back to the **File Inclusion** section and capture the request in your burpsuite.

```
GET /dvwa/vulnerabilities/fi/?page=file1.php HTTP/1.1
Host: 192.168.29.227
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.29.227/dvwa/vulnerabilities/fi/?page=include.php
Connection: close
Cookie: security=high; PHPSESSID=v1816gji96h08p8vadcf8k7nhb
Upgrade-Insecure-Requests: 1
```

Share the HTTP request to the repeater tab and manipulate the URL page parameter with:

```
page=file://C:/Windows/win.ini
```

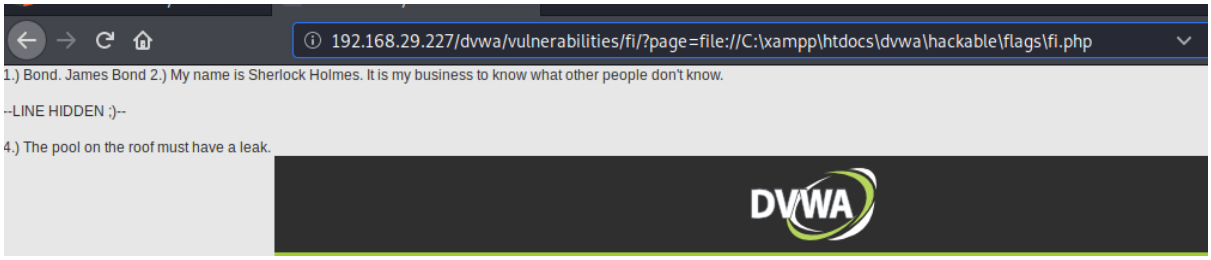
From the below image you can see that we have captured the **"win.ini"** file by entering the *complete path* to it in the URL parameter.




Let's now try to capture the **flag** with the same procedure as :

```
page=file://C:/xampp\htdocs\dvwa\hackable\flags\fi.php
```

Great!! We have grabbed this **hackable flag** too**.**




# Mitigation Steps

1. The developer should create a **whitelist** of all the files that he wants to include in order to limit the attacker's control.
2. Develop or run the code in the most recent version of the webserver which is available. The web applications should even be implemented with the least privileges.
3. Exclusion of the directory separators **"/"** to prevent the web application from the directory traversal attacks

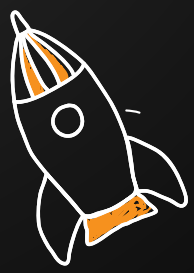
iGNITE Technologies

# JOIN OUR TRAINING PROGRAMS

CLICK HERE

## BEGINNER

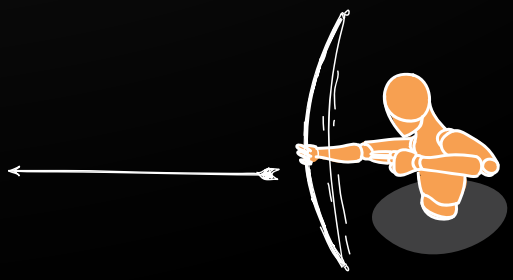
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



## ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

## EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux