



Metasploit

A Pentester Guide



Table of Contents

Abstract.....	4
Introduction.....	5
Creating a Workspace	6
Database Nmap Scan	6
Hosts Database	9
Services Database	11
Vulnerabilities Database.....	13
Credentials Database	14
Renaming Database	15
Deleting Database	15
Verbose Details of Workspaces.....	16
Exporting Workspace	17
Deleting all Workspaces.....	17
Importing Hosts	18
Importing Hosts	18
Adding Hosts.....	19
Deleting a host	20
Exporting Hosts Database.....	21
Search and Tag Hosts.....	21
Filter Search.....	22
Deleting Services of a Host.....	23
Filter Search.....	24
Port Specific Services	25
Exporting Services Database.....	25
Search Services	26
Port Specific Vulnerabilities	27
Service Specific Vulnerabilities	27
Host Specific Vulnerabilities	28
Exporting Vulnerabilities Database	29
Loot Database	29



Search Loot..... 31

Type Specific Loot 31

Deleting Loot..... 32

Conclusion 33

References 33



Abstract

In this report, we are focusing on the various mechanisms of the Metasploit Framework that can be used by Penetration Testers. We are going to learn about the workspace and database commands of the Metasploit Framework.

Disclaimer: This report is provided for educational and informational purpose only (Penetration Testing). Penetration Testing refers to legal intrusion tests that aim to identify vulnerabilities and improve cybersecurity, rather than for malicious purposes.



Introduction

The database service provided by Metasploit is one of the greatest for keeping a record of your Penetration Testing activity, one of the greatest things about the database is the fact that if you have a bigger assessment then you have more reasons to use the database and its features. It keeps a record of all the hosts that you have tracked and gathered data from. It also records the time and date as of when you enumerated and gathered data so that you can keep a track of your activity. To initiate the PostgreSQL database, use the command `msfdb init` to initialize the database.

```
msfdb init
```

```
(root@kali)-[~]  
# msfdb init  
[+] Starting database  
[i] The database appears to be already configured, skipping initialization
```

After you initialized the database, you can check the status to verify if the database was initialized successfully using the `db_status` command. For a session you don't need to initialize the database again.

```
db_status
```

```
msf6 > db_status  
[*] Connected to msf. Connection type: postgresql.  
msf6 >
```

Using Workspaces helps you to divide and structure your Penetration Testing assessments. It helps by managing the hosts and data derived from those hosts in a database. Workspaces can be used to logically separate targets and assessments. For example, if you require to create a workspace for each subnet within an organization that you are performing Penetration Testing on. It will restrict the hosts to a specific network in a separate workspace which will be easy to understand and manage.

```
workspace -h
```



```
msf6 > workspace -h
Usage:
workspace                List workspaces
workspace -v             List workspaces verbosely
workspace [name]         Switch workspace
workspace -a [name] ...  Add workspace(s)
workspace -d [name] ...  Delete workspace(s)
workspace -D             Delete all workspaces
workspace -r <old> <new> Rename workspace
workspace -h             Show this help information
```

Creating a Workspace

Now that we have discussed in detail the Database command and the Workspace in the Metasploit Framework, it is time to begin the demonstration of the various options and actions that can be performed using them. Working with the workspace, we have the default workspace that as you might have guessed it comes and is enabled by default. It is possible to create your own workspace. This can help you collect the data based on the different projects as the penetration tester. For the demonstration, we will create a workspace by the name of pentest. By simply running the workspace without any option will list the various workspaces present in the database. Workspace with an asterisk (*) denotes the current workspace.

```
workspace -a pentest
```

```
msf6 > workspace -a pentest
[*] Added workspace: pentest
[*] Workspace: pentest
msf6 > workspace
default
* pentest
msf6 >
```

Database Nmap Scan

As a part of penetration testing any machine, it is fundamental to perform a port scan. Earlier we used to perform a Nmap port scan and then export the result into a file for future reference. But with the help of workspace, we can perform a Nmap scan and save its result into the workspace database. From the Metasploit shell, we need to run the db_nmap



command with the usual Nmap options to run a Nmap scan and save its result into the workspace database. In the demonstration, we performed the Nmap scan against the entire subnet. From all the active machines that have been detected by Nmap, we selected the two targets 192.168.1.12 and 192.168.1.16 for our assessment.

```
db_nmap -sP 192.168.1.0/24
```

```
msf6 > db_nmap -sP 192.168.1.0/24
[*] Nmap: Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-04 12:45 EDT
[*] Nmap: Nmap scan report for dsldevice.lan (192.168.1.1)
[*] Nmap: Host is up (0.0014s latency).
[*] Nmap: MAC Address: 18:45:93:69:A5:10 (Taicang T&W Electronics)
[*] Nmap: Nmap scan report for 192.168.1.3
[*] Nmap: Host is up (0.00011s latency).
[*] Nmap: MAC Address: 8C:EC:4B:71:C5:DE (Dell)
[*] Nmap: Nmap scan report for 192.168.1.4
[*] Nmap: Host is up (0.040s latency).
[*] Nmap: MAC Address: 2A:84:98:9F:E5:5E (Unknown)
[*] Nmap: Nmap scan report for 192.168.1.6
[*] Nmap: Host is up (0.13s latency).
[*] Nmap: MAC Address: 44:CB:8B:C2:20:DA (LG Innotek)
[*] Nmap: Nmap scan report for 192.168.1.12
[*] Nmap: Host is up (0.00035s latency).
[*] Nmap: MAC Address: 00:0C:29:78:20:90 (VMware)
[*] Nmap: Nmap scan report for 192.168.1.15
[*] Nmap: Host is up (0.17s latency).
[*] Nmap: MAC Address: 38:A4:ED:CF:8E:8D (Xiaomi Communications)
[*] Nmap: Nmap scan report for 192.168.1.16
[*] Nmap: Host is up (0.00011s latency).
[*] Nmap: MAC Address: 00:0C:29:5C:69:16 (VMware)
[*] Nmap: Nmap scan report for 192.168.1.9
[*] Nmap: Host is up.
[*] Nmap: Nmap done: 256 IP addresses (8 hosts up) scanned in 21.27 seconds
```

Since we have the selected target, we can perform a focused Aggressive Nmap scan against the target host. In the demonstration, we are targeting the host with the IP Address 192.168.1.12. We see that the options and the result that is used and generated are quite similar to the one we use normally with the difference that all the enumeration that is being done is being saved in the workspace database that we created earlier.

```
db_nmap -A 192.168.1.12
```




```
msf6 > db_nmap -A 192.168.1.12
[*] Nmap: Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-04 12:53 EDT
[*] Nmap: Nmap scan report for 192.168.1.12
[*] Nmap: Host is up (0.0012s latency).
[*] Nmap: Not shown: 977 closed ports
[*] Nmap: PORT      STATE SERVICE      VERSION
[*] Nmap: 21/tcp    open  ftp          vsftpd 2.3.4
[*] Nmap: |_ftp-anon: Anonymous FTP login allowed (FTP code 230)
[*] Nmap: |_ftp-syst:
[*] Nmap:   STAT:
[*] Nmap:   FTP server status:
[*] Nmap:     Connected to 192.168.1.9
[*] Nmap:     Logged in as ftp
[*] Nmap:     TYPE: ASCII
[*] Nmap:     No session bandwidth limit
[*] Nmap:     Session timeout in seconds is 300
[*] Nmap:     Control connection is plain text
[*] Nmap:     Data connections will be plain text
[*] Nmap:     vsFTPD 2.3.4 - secure, fast, stable
[*] Nmap: |_End of status
[*] Nmap: 22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
[*] Nmap: |_ssh-hostkey:
[*] Nmap:   1024 60:0f:cf:e1:c0:5f:6a:74:d6:90:24:fa:c4:d5:6c:cd (DSA)
[*] Nmap:   2048 56:56:24:0f:21:1d:de:a7:2b:ae:61:b1:24:3d:e8:f3 (RSA)
[*] Nmap: 23/tcp    open  telnet       Linux telnetd
[*] Nmap: 25/tcp    open  smtp         Postfix smtpd
[*] Nmap: |_smtp-commands: metasploitable.localdomain, PIPELINING, SIZE 1024000
[*] Nmap: |_ssl-date: 2021-07-04T16:53:47+00:00; -7s from scanner time.
[*] Nmap:   sslv2:
[*] Nmap:     SSLv2 supported
[*] Nmap:     ciphers:
[*] Nmap:       SSL2_RC2_128_CBC_EXPORT40_WITH_MD5
[*] Nmap:       SSL2_DES_64_CBC_WITH_MD5
[*] Nmap:       SSL2_RC4_128_WITH_MD5
[*] Nmap:       SSL2_RC2_128_CBC_WITH_MD5
[*] Nmap:       SSL2_DES_192_EDE3_CBC_WITH_MD5
[*] Nmap:       SSL2_RC4_128_EXPORT40_WITH_MD5
[*] Nmap: 53/tcp    open  domain       ISC BIND 9.4.2
[*] Nmap: |_dns-nsid:
[*] Nmap:   |_bind.version: 9.4.2
[*] Nmap: 80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
[*] Nmap: |_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
[*] Nmap: |_http-title: Metasploitable2 - Linux
[*] Nmap: 111/tcp   open  rpcbind      2 (RPC #100000)
[*] Nmap: |_rpcinfo:
[*] Nmap:   program version  port/proto  service
[*] Nmap:   100000  2          111/tcp    rpcbind
[*] Nmap:   100000  2          111/udp    rpcbind
[*] Nmap:   100003  2,3,4      2049/tcp   nfs
[*] Nmap:   100003  2,3,4      2049/udp   nfs
[*] Nmap:   100005  1,2,3      33096/udp  mountd
[*] Nmap:   100005  1,2,3      51416/tcp  mountd
[*] Nmap:   100021  1,3,4      48805/udp  nlockmgr
[*] Nmap:   100021  1,3,4      59009/tcp  nlockmgr
[*] Nmap:   100024  1          36091/udp  status
[*] Nmap:   100024  1          52208/tcp  status
```




Hosts Database

As we used the `db_nmap` for scanning the host 192.168.1.12, after the conclusion of the scan we can run the `hosts` command. It will list all the hosts that have been subject to the port scan performed using the `db_nmap`. We can see that our Nmap scan was able to get the MAC Address, Operating System and Purpose for the targeted host. We can use a list of different scan options and auxiliary tools to enumerate and the data that is collected will automatically be filled inside the table presented.

hosts

```
msf6 > hosts   
  
Hosts  
====  
  
address      mac          name  os_name  os_flavor  os_sp  purpose  
-----  
192.168.1.12  00:0c:29:78:20:90  Linux 2.6.X    server
```

Moving on, the `db_nmap` command is not the only method to add the data inside the workspace data. There are various specifications of different organisations that demand that the penetration tester uses Nmap and export the scan result into a predefined format. If that is the case with you, it is still possible to use the workspace database. Let's us demonstrate. First, we will be performing a Version Nmap scan on our other targeted hosts 192.168.1.16 with the `-oX` option so that the scan result is exported into an XML file.

nmap -sV -oX 192.168.1.16



```
(root@kali)-[~]
# nmap -sV -oX result 192.168.1.16
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-04 12:52 EDT
Nmap scan report for 192.168.1.16
Host is up (0.00081s latency).
Not shown: 991 closed ports
PORT      STATE SERVICE      VERSION
135/tcp    open  msrpc        Microsoft Windows RPC
139/tcp    open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp    open  microsoft-ds Microsoft Windows 7 - 10 microsoft
49152/tcp  open  msrpc        Microsoft Windows RPC
49153/tcp  open  msrpc        Microsoft Windows RPC
49154/tcp  open  msrpc        Microsoft Windows RPC
49155/tcp  open  msrpc        Microsoft Windows RPC
49157/tcp  open  msrpc        Microsoft Windows RPC
49158/tcp  open  msrpc        Microsoft Windows RPC
MAC Address: 00:0C:29:5C:69:16 (VMware)
Service Info: Host: WIN-3Q7NEBI2561; OS: Windows; CPE: cpe:/o:
```

We have the Nmap scan result saved into an XML file by the name of the result. We will use the db_import command from the Metasploit shell to import the data from the result file. After exporting the data, we ran the hosts command to find that the data from the external Nmap scan is now a part of the Workspace database.

```
db_import result
```

```
hosts
```

```
msf6 > db_import result
[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.11.7'
[*] Importing host 192.168.1.16
[*] Successfully imported /root/result
msf6 > hosts

Hosts
=====

address      mac              name  os_name  os_flavor  os_sp  purpose
-----
192.168.1.12 00:0c:29:78:20:90 Linux      2.6.X    server
192.168.1.16 00:0c:29:5c:69:16 Unknown    device
```




Services Database

Now we have two hosts, 192.168.1.12 and 192.168.1.16. We have performed db_nmap and Nmap scan on them respectively. We saw earlier that we have the information about both machines in the hosts table but we know one of the primary goals of performing a Nmap scan is to enumerate the open ports and running services on the target machine. Since we used the db_nmap scan and imported a Nmap scan, we should have the details about the services running on both machines. We can check them out using the services command as demonstrated. To get a defined result about a particular result uses the IP address of your target as a parameter while running the services command.


```
services
```

```
services 192.168.1.12
```



```
msf6 > services   
Services  
=====
```

host	port	proto	name	state	info
192.168.1.12	21	tcp	ftp	open	vsftpd 2.3.4
192.168.1.12	22	tcp	ssh	open	OpenSSH 4.7p1 Debian 8ubuntu
192.168.1.12	23	tcp	telnet	open	Linux telnetd
192.168.1.12	25	tcp	smtp	open	Postfix smtpd
192.168.1.12	53	tcp	domain	open	ISC BIND 9.4.2
192.168.1.12	80	tcp	http	open	Apache httpd 2.2.8 (Ubuntu)
192.168.1.12	111	tcp	rpcbind	open	2 RPC #100000
192.168.1.12	139	tcp	netbios-ssn	open	Samba smbd 3.X - 4.X workgr
192.168.1.12	445	tcp	netbios-ssn	open	Samba smbd 3.0.20-Debian wo
192.168.1.12	512	tcp	exec	open	netkit-rsh rexecd
192.168.1.12	513	tcp	login	open	OpenBSD or Solaris rlogind
192.168.1.12	514	tcp	tcpwrapped	open	
192.168.1.12	1099	tcp	java-rmi	open	GNU Classpath grmiregistry
192.168.1.12	1524	tcp	bindshell	open	Metasploitable root shell
192.168.1.12	2049	tcp	nfs	open	2-4 RPC #100003
192.168.1.12	2121	tcp	ftp	open	ProFTPD 1.3.1
192.168.1.12	3306	tcp	mysql	open	MySQL 5.0.51a-3ubuntu5
192.168.1.12	5432	tcp	postgresql	open	PostgreSQL DB 8.3.0 - 8.3.7
192.168.1.12	5900	tcp	vnc	open	VNC protocol 3.3
192.168.1.12	6000	tcp	x11	open	access denied
192.168.1.12	6667	tcp	irc	open	UnrealIRCd
192.168.1.12	8009	tcp	ajp13	open	Apache Jserv Protocol v1.3
192.168.1.12	8180	tcp	http	open	Apache Tomcat/Coyote JSP en
192.168.1.16	135	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	139	tcp	netbios-ssn	open	Microsoft Windows netbios-s
192.168.1.16	445	tcp	microsoft-ds	open	Microsoft Windows 7 - 10 mi
192.168.1.16	49152	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49153	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49154	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49155	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49157	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49158	tcp	msrpc	open	Microsoft Windows RPC

```
msf6 > services 192.168.1.16   
Services  
=====
```

host	port	proto	name	state	info
192.168.1.16	135	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	139	tcp	netbios-ssn	open	Microsoft Windows netbios-s
192.168.1.16	445	tcp	microsoft-ds	open	Microsoft Windows 7 - 10 mi
192.168.1.16	49152	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49153	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49154	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49155	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49157	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.16	49158	tcp	msrpc	open	Microsoft Windows RPC



Vulnerabilities Database

After the detection of hosts and the running services, we need to find vulnerabilities on the target machine. We will be using the db_nmap to scan the target hosts with a Vulnerability Script scan. It will help to enumerate the target with possible vulnerabilities. We will be targeting the SSH service running on 192.168.1.12.

```
db_nmap -sV -p22 --script=vuln 192.168.1.12
```

```
msf6 > db_nmap -sV -p22 --script=vuln 192.168.1.12
[*] Nmap: Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-04 13:04 EDT
[*] Nmap: Pre-scan script results:
[*] Nmap: | broadcast-avahi-dos:
[*] Nmap: |   Discovered hosts:
[*] Nmap: |     224.0.0.251
[*] Nmap: |   After NULL UDP avahi packet DoS (CVE-2011-1002).
[*] Nmap: | _ Hosts are all up (not vulnerable).
[*] Nmap: Nmap scan report for 192.168.1.12
[*] Nmap: Host is up (0.00065s latency).
[*] Nmap: PORT      STATE SERVICE VERSION
[*] Nmap: 22/tcp open  ssh      OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
[*] Nmap: | vulners:
[*] Nmap: |   cpe:/a:openbsd:openssh:4.7p1:
[*] Nmap: |     EDB-ID:21018    10.0    https://vulners.com/exploitdb/EDB-ID:21018
[*] Nmap: |     CVE-2001-0554   10.0    https://vulners.com/cve/CVE-2001-0554
[*] Nmap: |     PACKETSTORM:105078 7.8     https://vulners.com/packetstorm/PACKETSTORM:105078
[*] Nmap: |     PACKETSTORM:101052 7.8     https://vulners.com/packetstorm/PACKETSTORM:101052
[*] Nmap: |     SECURITYVULNS:VULN:8166 7.5     https://vulners.com/securityvulns/SECURITYVULNS:VULN:8166
[*] Nmap: |     MSF:ILITIES/OPENBSD-OPENSSSH-CVE-2010-4478/ 7.5     https://vulners.com/msf/ILITIES/OPENBSD-OPENSSSH-CVE-2010-4478/
[*] Nmap: |     MSF:ILITIES/LINUXRPM-ELSA-2008-0855/ 7.5     https://vulners.com/msf/ILITIES/LINUXRPM-ELSA-2008-0855/
[*] Nmap: |     CVE-2010-4478   7.5     https://vulners.com/cve/CVE-2010-4478
[*] Nmap: |     SSV:20512       7.2     https://vulners.com/seebug/SSV:20512
[*] Nmap: |     CVE-2011-1013   7.2     https://vulners.com/cve/CVE-2011-1013
[*] Nmap: |     CVE-2008-1657   6.5     https://vulners.com/cve/CVE-2008-1657
[*] Nmap: |     SSV:60656       5.0     https://vulners.com/seebug/SSV:60656
[*] Nmap: |     CVE-2017-15906   5.0     https://vulners.com/cve/CVE-2017-15906
[*] Nmap: |     CVE-2011-2168   5.0     https://vulners.com/cve/CVE-2011-2168
[*] Nmap: |     CVE-2010-5107   5.0     https://vulners.com/cve/CVE-2010-5107
[*] Nmap: |     CVE-2007-2768   4.3     https://vulners.com/cve/CVE-2007-2768
[*] Nmap: |     CVE-2010-4754   4.0     https://vulners.com/cve/CVE-2010-4754
```

After the completion of the Vulnerability Script scan using the db_nmap, we can check for the vulnerabilities detected using the command vulns as demonstrated below. Here we have a table with the Timestamp when the vulnerability was detected, with the Host on which the vulnerability was detected, the name of the vulnerability and the Reference containing the respective CVEs and EDB details.



vulns

```
msf6 > vulns
Vulnerabilities
=====
Timestamp      Host      Name      References
-----
2021-07-04 17:04:51 UTC 192.168.1.12 cpe:/a:openbsd:openssh:4.7p1 EDB-ID:21018,CVE-2001-0554,PACKE
5/,CVE-2010-4478,SSV:20512,CVE-2
/,MSF:ILITIES/ORACLE-SOLARIS-CVE
61,CVE-2011-4327,MSF:ILITIES/SSH
```

Credentials Database

From vulnerabilities, we move on to the extraction of valuable credentials from the target machine and saving them to the workspace database. When you use any Metasploit module that extracts or attempts credentials on the target machine then it gets saved inside the credentials section. This section can be viewed by running the creds command. In the demonstration, we use the ftp_login auxiliary scanner to perform a Bruteforce on the FTP login on our target machine. We can see the credentials that were correct have made an entry inside the Credentials section.

```
msf6 > use auxiliary/scanner/ftp/ftp_login
msf6 auxiliary(scanner/ftp/ftp_login) > set rhosts 192.168.1.40
rhosts => 192.168.1.40
msf6 auxiliary(scanner/ftp/ftp_login) > set user_file /root/users.txt
user_file => /root/users.txt
msf6 auxiliary(scanner/ftp/ftp_login) > set pass_file /root/pass.txt
pass_file => /root/pass.txt
msf6 auxiliary(scanner/ftp/ftp_login) > set stop_on_success true
stop_on_success => true
msf6 auxiliary(scanner/ftp/ftp_login) > set verbose false
verbose => false
msf6 auxiliary(scanner/ftp/ftp_login) > exploit

[*] 192.168.1.40:21 - 192.168.1.40:21 - Starting FTP login sweep
[+] 192.168.1.40:21 - 192.168.1.40:21 - Login Successful: privs:123
[*] 192.168.1.40:21 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ftp/ftp_login) > back
msf6 > creds
Credentials
=====
host      origin      service      public      private      realm      private_type      JtR Format
-----
192.168.1.40 192.168.1.40 21/tcp (ftp) privs      123      Password
```

After the brief introduction of the various command that can be run inside the Metasploit shell, we will now focus on the various aspects of them in detail. Starting from Workspace.



Renaming Database

Earlier we learnt about creating a workspace but in case you require to rename a Workspace to provide better management it is possible with the help of the -r option. We first use the workspace command to list the various workspace. We see that there are three workspaces namely default, pentest and ignite. We will use the -r option to rename the pentest workspace to raj. And we can see that we were able to change the name of the said workspace.

```
workspace
```

```
workspace -r pentest raj
```

```
workspace
```

```
msf6 > workspace  
default  
pentest  
* ignite  
msf6 > workspace -r pentest raj  
[*] Renamed workspace 'pentest' to 'raj'  
msf6 > workspace  
default  
raj  
* ignite
```

Deleting Database

Now that we renamed a workspace, there might be a workspace that you want to delete from the database. Although keep in mind that deleting the database will delete all the consiquent data as well such as the hosts, vulns and loot from that database. In the scenario presented below, we are shown that there exist three workspaces namely default, raj and ignite. Using the -d option we deleted the ignite workspace.



`workspace`

`workspace -d ignite`

`workspace`

```
msf6 > workspace  
default  
raj  
* ignite  
msf6 > workspace -d ignite  
[*] Deleted workspace: ignite  
[*] Switched to workspace: default  
msf6 > workspace  
raj  
* default
```

Verbose Details of Workspaces

During the penetration testing, there will be a time where you might require to get a quick look at your various workspaces. When you make dedicated workspaces for different projects and need a reference for the data stored inside the workspace such as the hosts detected, services running, vulnerabilities found, credentials scrapped and loots grabbed. Use the `-v` option to check out all this information at once.

`workspace -v`

```
msf6 > workspace -v  
  
Workspaces  
=====
```

current	name	hosts	services	vulns	creds	loots	notes
*	default	0	0	0	0	0	0
	raj	3	33	1	1	0	31



Exporting Workspace

As we discussed earlier that documentation is an important part of penetration testing. While working with workspace it gets easier to sort your data but with the ability to export the data from the workspace into an XML file for creating reports is an underrated feature. Use the `db_export` command for this task. You will need to provide the format you want to use with the `-f` option followed by the name of the file as demonstrated below.

```
db_export -f xml workspace_backup.xml
```

```
msf6 > db_export -f xml workspace_backup.xml
[*] Starting export of workspace raj to workspace_backup.xml [ xml ] ...
[*] Finished export of workspace raj to workspace_backup.xml [ xml ] ...
msf6 > 
```

Deleting all Workspaces

Every once a while there comes a time when even the very structured format of the workspace becomes a mess-making no sense at all. You could always delete the workspace as we previously demonstrated. But in case you have multiple workspaces and now want to use the nuclear option on all of them then `-D` option comes in handy.

```
workspace -D
```

```
msf6 > workspace -D
[*] Deleted workspace: default
[*] Recreated the default workspace
[*] Deleted workspace: raj
[*] Switched to workspace: default
msf6 > workspace
* default
```



Importing Hosts

Just earlier we exported the data from a workspace into an XML file. We mentioned that this file can be used for references and creating reports but this XML can also serve as a backup for your work. It is highly unlikely that your works gets deleted or corrupted from the workspace database. We tried to do but we were not able to kill the database. We even restarted the system to check but all the data seemed to retain inside the workspace. However, if you want to import the data from your workspace backup then you can use the `db_import` command.

```
db_import workspace_backup.xml
```

```
hosts
```

```
msf6 > db_import workspace_backup.xml
[*] Importing 'Metasploit XML' data
[*] Importing host 192.168.1.16
[*] Importing host 192.168.1.12
[*] Importing host 192.168.1.40
[*] Successfully imported /root/workspace_backup.xml
msf6 > hosts

Hosts
=====

address      mac                name  os_name  os_flavor  os_sp
-----
192.168.1.12  00:0c:29:78:20:90   Linux
192.168.1.16  00:0c:29:5c:69:16   Unknown
192.168.1.40
```

This completes our testing with the workspace commands. Let's move on to the different options inside the host's command.

Importing Hosts

As we saw earlier that when we perform a Nmap scan with the `db_nmap` command we populate the host's table. We can list all the hosts and the data that was enumerated from the



Nmap scan into a structured table with the IP Address, MAC Address, Name, Operating System and other details.

hosts

```
msf6 > hosts
```

Hosts

address	mac	name	os_name	os_flavor	os_sp	purpose
192.168.1.1	18:45:93:69:a5:10	dsldevice.lan	Linux		3.X	server
192.168.1.3	8c:ec:4b:71:c5:de		Windows XP			client
192.168.1.4	2a:84:98:9f:e5:5e		Linux		2.6.X	server
192.168.1.6	44:cb:8b:c2:20:da		Linux		3.X	server
192.168.1.12	00:0c:29:78:20:90		Linux		2.6.X	server
192.168.1.16	00:0c:29:5c:69:16		Windows 7			client
192.168.1.40	00:0c:29:c4:86:93		Linux		4.X	server
192.168.1.101						
192.168.1.102						

Adding Hosts

Earlier we used the Nmap scan to enumerate for hosts inside the database. But it is not the only way to do so. We can add hosts using the -a option as well. In scenarios where you want to add particular hosts into your database so that you can perform attacks from the hosts table then you can add hosts as demonstrated below:

```
hosts -a 192.168.1.101 192.168.1.102
```



```
msf6 > hosts -a 192.168.1.101 192.168.1.102
[*] Time: 2021-07-04 18:50:14 UTC Host: host=192.168.1.101
[*] Time: 2021-07-04 18:50:14 UTC Host: host=192.168.1.102
msf6 > hosts

Hosts
=====
```

address	mac	name	os_name	os_flavor	os_sp	purpose
192.168.1.1	18:45:93:69:a5:10	dsldevice.lan	Linux		3.X	server
192.168.1.3	8c:ec:4b:71:c5:de		Windows XP			client
192.168.1.4	2a:84:98:9f:e5:5e		Linux		2.6.X	server
192.168.1.6	44:cb:8b:c2:20:da		Linux		3.X	server
192.168.1.12	00:0c:29:78:20:90		Linux		2.6.X	server
192.168.1.16	00:0c:29:5c:69:16		Windows 7			client
192.168.1.40	00:0c:29:c4:86:93		Linux		4.X	server
192.168.1.100						
192.168.1.101						
192.168.1.102						

Deleting a host

As we can add a host into the database it is also possible to remove or delete any user from the database. This can be achieved using the -d option. Previously we added two IP addresses into the database. Now we will delete one of them.

```
hosts -d 192.168.1.102
```

```
msf6 > hosts -d 192.168.1.102

Hosts
=====
```

address	mac	name	os_name	os_flavor	os_sp	purpose	info	comments
192.168.1.102								

```
[*] Deleted 1 hosts
```

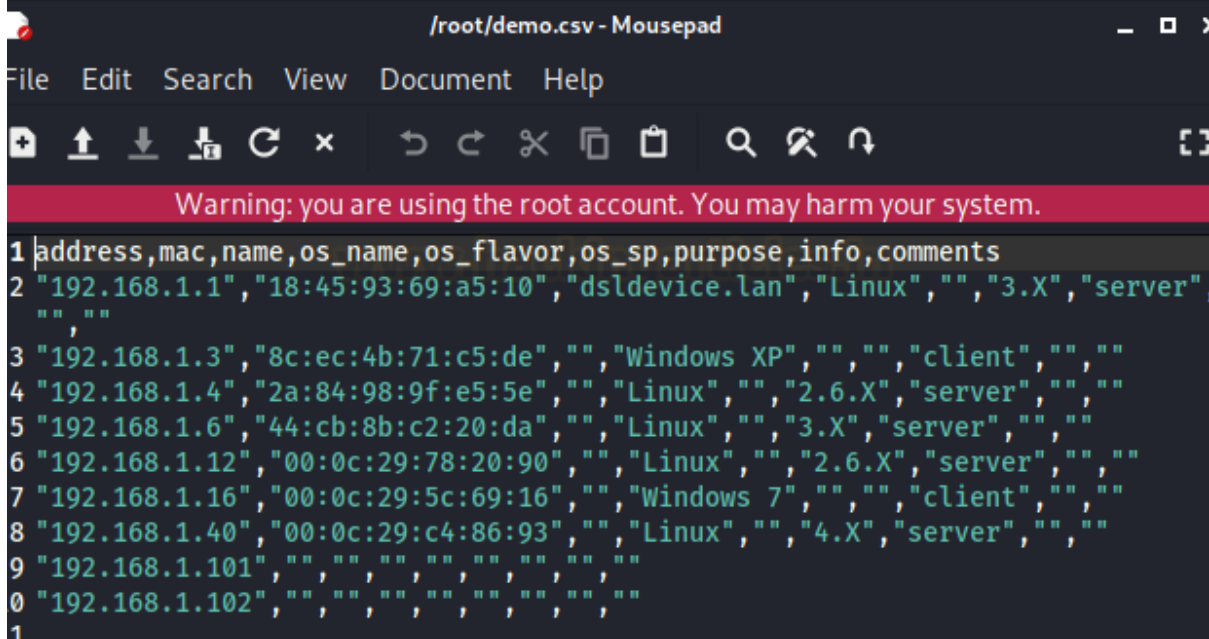


Exporting Hosts Database

As we mentioned before that the most important part of any penetration testing activity is the documentation process. We were able to export the workspace data into an XML file earlier. But in case you want the table of the hosts populated in your database into a manageable CSV file it can be done using the `-o` option. This CSV file can then be used for uploading it into a Scanner or Software as it contains all your targeted IP Addresses.

```
hosts -o demo.csv
```

```
msf6 > hosts -o demo.csv
[*] Wrote hosts to demo.csv
```



address	mac	name	os_name	os_flavor	os_sp	purpose	info	comments
"192.168.1.1"	"18:45:93:69:a5:10"	"dsldevice.lan"	"Linux"	"3.X"	"server"			
"192.168.1.3"	"8c:ec:4b:71:c5:de"		"Windows XP"		"client"			
"192.168.1.4"	"2a:84:98:9f:e5:5e"		"Linux"	"2.6.X"	"server"			
"192.168.1.6"	"44:cb:8b:c2:20:da"		"Linux"	"3.X"	"server"			
"192.168.1.12"	"00:0c:29:78:20:90"		"Linux"	"2.6.X"	"server"			
"192.168.1.16"	"00:0c:29:5c:69:16"		"Windows 7"		"client"			
"192.168.1.40"	"00:0c:29:c4:86:93"		"Linux"	"4.X"	"server"			
"192.168.1.101"								
"192.168.1.102"								

Search and Tag Hosts

While working with multiple hosts inside a dense network of machines it becomes difficult to identify and search for a particular target. Hosts command has the `-S` option that can help you search for a particular machine on the basis of a keyword such as the Operating System of the machine. In the demonstration, we used the keyword Windows. This gave us the two machines that were detected running Windows Operating System. Similarly, it is also



possible to define a tag for your hosts to make them easily identifiable. In the demonstration below, we have tagged the system bearing the IP Address 192.168.1.3 as raj-pc. We then can use this tag to search for the device in question.

```
hosts -S Windows
```

```
hosts -t raj-pc 192.168.1.3
```

```
hosts -S raj-pc
```

```
msf6 > hosts -S Windows
Hosts
=====
address      mac          name  os_name  os_flavor  os_sp  purpose
-----
192.168.1.3   8c:ec:4b:71:c5:de  Windows XP
192.168.1.16 00:0c:29:5c:69:16  Windows 7   client
192.168.1.16 00:0c:29:5c:69:16  Windows 7   client

msf6 > hosts -t raj-pc 192.168.1.3
msf6 > hosts -S raj-pc
Hosts
=====
address      mac          name  os_name  os_flavor  os_sp  purpose
-----
192.168.1.3   8c:ec:4b:71:c5:de  Windows XP   client
```



Filter Search

As we observed in the previous steps that the hosts table contains a variety of columns. But it is possible to view only the columns that you desire in your host's table. It can be done with the help of a filtered search. In the demonstration we used -c option to filter only the address, mac and os_name columns from the hosts table and another example with only IP Address and the Operating System for a clean and readable format.

```
hosts -c address,mac,os_name
```



```
hosts -c address,os_name
```

```
msf6 > hosts -c address,mac,os_name   
  
Hosts  
====  
  
address      mac          os_name  
-----  
192.168.1.1  18:45:93:a5:10 Linux  
192.168.1.3  8c:ec:4b:71:c5:de Windows XP  
192.168.1.4  2a:84:98:9f:e5:5e Linux  
192.168.1.6  44:cb:8b:c2:20:da Linux  
192.168.1.12 00:0c:29:78:20:90 Linux  
192.168.1.16 00:0c:29:5c:69:16 Windows 7  
192.168.1.40 00:0c:29:c4:86:93 Linux  
192.168.1.101  
192.168.1.102  
  
msf6 > hosts -c address,os_name   
  
Hosts  
====  
  
address      os_name  
-----  
192.168.1.1  Linux  
192.168.1.3  Windows XP  
192.168.1.4  Linux  
192.168.1.6  Linux  
192.168.1.12 Linux  
192.168.1.16 Windows 7  
192.168.1.40 Linux  
192.168.1.101  
192.168.1.102
```

Deleting Services of a Host

We introduced the services database earlier. It contained the various services running on the target machines that Nmap enumerated and saved in the services database. We can delete a host and all of their services along with it with the -d option as demonstrated below.

```
services -d 192.168.1.40
```



```
msf6 > services -d 192.168.1.40
Services
=====
```

host	port	proto	name	state	info
192.168.1.40	21	tcp	ftp	open	
192.168.1.40	22	tcp	ssh	open	
192.168.1.40	80	tcp	http	open	
192.168.1.40	139	tcp	netbios-ssn	open	
192.168.1.40	445	tcp	microsoft-ds	open	

```
[*] Deleted 5 services
```

Filter Search

The option to filter the columns in a workspace is not limited to the hosts command. We can use the `-c` option in the services as well. By default, there are a bunch of columns visible when we list the services in the database. With the help of `-c` option, we can filter our columns such as the Host, Port and Name of the Services as demonstrated below. As Host being the primary key in this table, it will display by default.

```
services -c port,name
```

```
msf6 > services -c port,name
Services
=====
```

host	port	name
192.168.1.1	80	http
192.168.1.1	443	https
192.168.1.3	135	msrpc
192.168.1.3	139	netbios-ssn
192.168.1.3	443	https
192.168.1.3	445	microsoft-ds
192.168.1.3	902	iss-realsecure
192.168.1.3	912	apex-mesh
192.168.1.3	1688	nsjtp-data
192.168.1.3	3389	ms-wbt-server
192.168.1.3	5357	wsdapi
192.168.1.3	7070	realserver
192.168.1.4	49152	unknown
192.168.1.6	1503	imtc-mcs
192.168.1.6	3000	ppp
192.168.1.6	3001	nessus
192.168.1.6	11111	vce
192.168.1.12	21	ftp



Port Specific Services

When we deal with the services, the most common way to recognize and differentiate between services is based on the port numbers. As there are default ports for various services that can however be changed but if not we can use the -p option to make a list of all the hosts that have a certain service running on the port. It is not limited to a single port, we can use multiple ports and even a range of ports as well.

```
services -p 80
```

```
msf6 > services -p 80
Services
=====
```

host	port	proto	name	state	info
192.168.1.1	80	tcp	http	open	
192.168.1.12	80	tcp	http	open	

Exporting Services Database

Till now we have exported the complete workspace database and hosts database. But the ability to export the data is not limited to those commands. We can export the data in the services database into a CSV file similarly as we did with the hosts database.

```
services -o result.csv
```



```
msf6 > services -o result.csv
[*] Wrote services to /root/result.csv
```

```
1 host,port,proto,name,state,info
2 "192.168.1.12","21","tcp","ftp","open",""
3 "192.168.1.12","22","tcp","ssh","open",""
4 "192.168.1.12","23","tcp","telnet","open",""
5 "192.168.1.12","25","tcp","smtp","open",""
6 "192.168.1.12","53","tcp","domain","open",""
7 "192.168.1.12","80","tcp","http","open",""
8 "192.168.1.1","80","tcp","http","open",""
9 "192.168.1.12","111","tcp","rpcbind","open",""
10 "192.168.1.16","135","tcp","msrpc","open",""
11 "192.168.1.3","135","tcp","msrpc","open",""
```

Search Services

We can search services based on the hosts, ports, service names, their state and even with the Protocol they are running on. There are services running on the TCP and UDP and other protocols. We can search those services using the `-S` option. But it is not limited to protocol. You can use any keyword for the search.

```
services -S udp
```

```
msf6 > services -S udp
Services
=====
```

host	port	proto	name	state	info
192.168.1.16	137	udp	netbios-ns	unknown	
192.168.1.16	138	udp	netbios-dgm	unknown	
192.168.1.16	500	udp	isakmp	unknown	
192.168.1.16	1900	udp	upnp	unknown	
192.168.1.16	4500	udp	nat-t-ike	unknown	
192.168.1.16	5355	udp	llmnr	unknown	



Port Specific Vulnerabilities

Next, the database inside the workspace we saw was the database of the Vulnerabilities that are present in the target machine and are enumerated using Nmap script scan or any of the Metasploit Auxiliary scans. We can search for a specific vulnerability on the service running port. In the demonstration below we targeted port 22 which is famous for running SSH services and we can see that we have the various probable Vulnerabilities that were detected inside the References column.

```
vulns -p 22
```

```
msf6 > vulns -p 22
Vulnerabilities
=====
```

Timestamp	Host	Name	References
2021-07-04 19:28:21 UTC	192.168.1.12	cpe:/a:openbsd:openssh:4.7p1	EDB-ID:21018,CVE-2001-0555/,CVE-2010-4478,SSV:2051/,MSF:ILITIES/ORACLE-SOLA61,CVE-2011-4327,MSF:ILIT

Service Specific Vulnerabilities

We just listed the Vulnerabilities based on the port number but as we know that services are not always running on their default ports. We can target the particular service that we are looking for with the help of the -s option. Here, we presented all the ports from 1 to 65536 and the FTP as the service and we can see the various Vulnerabilities for the FTP service no matter on what port it is running on.

```
vulns -p 1-65536 -s ftp
```



```
msf6 > vulns -p 1-65536 -s ftp
Vulnerabilities
Timestamp      Host           Name           References
2021-07-04 19:37:17 UTC 192.168.1.12 cpe:/a:proftpd:proftpd:1.3.1 SSV:26016,SSV:24282,/SUSE-CVE-2019-18217
```

Host Specific Vulnerabilities

We are not limited to listing the Vulnerabilities data based on the port and service but we can also list the vulnerabilities that were detected in a particular host. This is one of the most useful representations and it provides a quick look at the services hosted on the targeted host and the possible vulnerabilities on it.

```
vulns -i 192.168.1.12
```

```
msf6 > vulns -i 192.168.1.12
Vulnerabilities
Timestamp      Host           Name           References
2021-07-04 19:28:21 UTC 192.168.1.12 cpe:/a:openbsd:openssh:4.7p1 EDB-ID:21018,CVE-2001-RPM-ELSA-2008-0855/,CV754,MSF:ILITIES/SUSE-CAS-2012-99/,CVE-2012-0259,SECURITYVULNS:VULN
2021-07-04 19:37:17 UTC 192.168.1.12 cpe:/a:isc:bind:9.4.2 SSV:2853,CVE-2008-0122NUXRPM-ELSA-2015-2658/ITPACK:D60DF5E24DE171D
```



Exporting Vulnerabilities Database

Similar to the hosts, services and workspace databases, we can export the data from the Vulnerabilities into a CSV file. Since the Vulnerabilities database table contains the hosts IP Address, Services running on the machine and the possible Vulnerabilities, it is basically a preliminary report that can be used during the Penetration Testing assessments.

```
vulns -o vulnerblities.csv
```

```
msf6 > vulns -o vulnerblities.csv
[*] Wrote vulnerability information to /root/vulnerblities.csv
```

```
1 Timestamp,Host,Name,References
2 "2021-07-04 19:28:21 UTC","192.168.1.12","cpe:/a:openbsd:openssh:-
4.7p1","EDB-ID:21018,CVE-2001-0554,PACKETSTORM:105078,PACKETSTORM:-
101052,SECURITYVULNS:VULN:8166,MSF:ILITIES/OPENBSD-OPENSSSH-
CVE-2010-4478/,MSF:ILITIES/LINUXRPM-ELSA-2008-0855/,CVE-2010-4478,SSV:-
20512,CVE-2011-1013,CVE-2008-1657,SSV:-
60656,CVE-2017-15906,CVE-2011-2168,CVE-2010-5107,CVE-2007-2768,CVE-2010-47
4,MSF:ILITIES/SUSE-CVE-2011-5000/,MSF:ILITIES/ORACLE-SOLARIS-
```

Loot Database

Up until this point we have demonstrated and discussed the various types of databases that are provided by the Metasploit Framework. That includes the Workspace database, Hosts database, Services database, Vulnerabilities database but we saved this one for the last. It is called the Loot database. During any Penetration Testing Assessments, there are times where you are able to exploit a vulnerability and get into the target. This is where you start some post-exploitation activities including enumerating for credentials and hashes. In case you are able to extract those using the Metasploit Post Exploitation Module, those will be stored inside the workspace that you are working on. To demonstrate, we have exploited and got a session on the Metasploitable Vulnerable Server and we will be using the enum_configs post-



exploitation module to extract the configurations files that might contain some passwords or important information stored in them.

```
msf6 > use post/linux/gather/enum_configs
msf6 post(linux/gather/enum_configs) > set session 1
session => 1
msf6 post(linux/gather/enum_configs) > exploit

[!] SESSION may not be compatible with this module (incompatible session platform: unix)
[*] Running module against 192.168.1.12 [metasploitable]
[*] Info:
[*]
[*]
[*]
[*] d network!Contact: msfdev[at]metasploit.comLogin with msfadmin/msfadmin to get started
[*] Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
[*] apache2.conf stored in /root/.msf4/loot/20210704154452_pentest_192.168.1.12_linux.enum.conf_494778.txt
[*] ports.conf stored in /root/.msf4/loot/20210704154453_pentest_192.168.1.12_linux.enum.conf_508192.txt
[*] my.cnf stored in /root/.msf4/loot/20210704154524_pentest_192.168.1.12_linux.enum.conf_838951.txt
[*] ufw.conf stored in /root/.msf4/loot/20210704154524_pentest_192.168.1.12_linux.enum.conf_018308.txt
[*] sysctl.conf stored in /root/.msf4/loot/20210704154524_pentest_192.168.1.12_linux.enum.conf_851507.txt
[*] shells stored in /root/.msf4/loot/20210704154540_pentest_192.168.1.12_linux.enum.conf_678618.txt
[*] access.conf stored in /root/.msf4/loot/20210704154611_pentest_192.168.1.12_linux.enum.conf_210675.txt
[*] rpc stored in /root/.msf4/loot/20210704154626_pentest_192.168.1.12_linux.enum.conf_721613.txt
[*] debian.cnf stored in /root/.msf4/loot/20210704154642_pentest_192.168.1.12_linux.enum.conf_779878.txt
[*] logrotate.conf stored in /root/.msf4/loot/20210704154657_pentest_192.168.1.12_linux.enum.conf_356901.txt
[*] smb.conf stored in /root/.msf4/loot/20210704154713_pentest_192.168.1.12_linux.enum.conf_505508.txt
[*] ldap.conf stored in /root/.msf4/loot/20210704154713_pentest_192.168.1.12_linux.enum.conf_279026.txt
[*] sysctl.conf stored in /root/.msf4/loot/20210704154800_pentest_192.168.1.12_linux.enum.conf_548520.txt
[*] Post module execution completed
```

As we can see that the enum_configs grabbed a bunch of config files. Now when we run the loot command on the Metasploit shell, we can see a detailed table of all the config files with the hosts that they were acquired from and the path at which they are currently stored as well.

loot

```
msf6 > loot

Loot
==

host      service  type      name      content  info  path
-----
192.168.1.12  linux.enum.conf  apache2.conf  text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  ports.conf    text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  my.cnf        text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  ufw.conf      text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  sysctl.conf   text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  shells       text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  access.conf   text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  rpc           text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  debian.cnf    text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  logrotate.conf  text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  smb.conf      text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  ldap.conf     text/plain  /root/.msf4/loot/
192.168.1.12  linux.enum.conf  sysctl.conf   text/plain  /root/.msf4/loot/
```



Search Loot

As it is possible to be working on multiple targets at a particular moment, the loot table might get populated too much with the data that it might be difficult to look for a particular loot that you might need at a given amount of time. We can use the Search option to search for a specific loot like we are searching for loot related to the ldap in the demonstration.

```
loot -S ldap
```

```
msf6 > loot -S ldap
Loot
=====
host      service  type      name      content    info      path
-----
192.168.1.12  linux.enum.conf ldap.conf  text/plain /root/.msf4/loot/20210704154711_
```

Type Specific Loot

As it is possible to be working on multiple targets, it is also possible that we are able to gather different types of loots based on the method used or exploit used on the target. We can sort and search for a particular type of loot with the help of -t option as demonstrated below.

```
loot -t linux.enum.conf
```



```
msf6 > loot -t linux.enum.conf

Loot
==
```

host	service	type	name	content	info	path
192.168.1.12		linux.enum.conf	apache2.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	ports.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	my.cnf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	ufw.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	sysctl.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	shells	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	access.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	rpc	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	debian.cnf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	logrotate.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	smb.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	ldap.conf	text/plain		/root/.msf4/loot/2
192.168.1.12		linux.enum.conf	sysctl.conf	text/plain		/root/.msf4/loot/2

Deleting Loot

As we did with each and every section and database discussed here, it is possible to delete the loot that you have acquired. To do so we can use the `-d` option. We will be using the keyword to target specific loot that we want to delete. For example, in case we want to delete the loot of a specific target, then we can provide the IP address of that target and delete all the loot from that source.

```
loot -d 192.168.1.12
```

```
msf6 > loot -d 192.168.1.12

Loot
==
```

host	service	type	name	content	i
192.168.1.12		linux.enum.conf	apache2.conf	text/plain	
192.168.1.12		linux.enum.conf	ports.conf	text/plain	
192.168.1.12		linux.enum.conf	my.cnf	text/plain	
192.168.1.12		linux.enum.conf	ufw.conf	text/plain	
192.168.1.12		linux.enum.conf	sysctl.conf	text/plain	
192.168.1.12		linux.enum.conf	shells	text/plain	
192.168.1.12		linux.enum.conf	access.conf	text/plain	
192.168.1.12		linux.enum.conf	rpc	text/plain	
192.168.1.12		linux.enum.conf	debian.cnf	text/plain	
192.168.1.12		linux.enum.conf	logrotate.conf	text/plain	
192.168.1.12		linux.enum.conf	smb.conf	text/plain	
192.168.1.12		linux.enum.conf	ldap.conf	text/plain	
192.168.1.12		linux.enum.conf	sysctl.conf	text/plain	

```
[*] Deleted 13 loots
```




Conclusion

This was a learning experience as when we start with the Penetration Activities, we tend not to focus on the documentation process or providing you work a proper structure and backup. But with time and some incidents where lack of these qualities proves to be valuable. The Workspace and the database function of Metasploit is not a new feature, it has been on for years and yet the general usage of these in the real life seems very less. Hence, it inspired us to provide the guide, so that lots of Penetration Testers can use it and benefit from it.

Hence, one can make use of these commands as a cybersecurity professional to assess vulnerabilities on systems and keep these systems away from threat.

References

- <https://www.hackingarticles.in/metasploit-for-pentester-database-workspace/>