iGNITE Technologies

Metasploit Framework

DATABASE & WORKSPACE

WWW.HACKINGARTICLES.IN

Contents

Introduction 4

Creating a Workspace 5

Database Nmap Scan 5

Hosts Database 7

Services Database 9

Vulnerabilities Database 10

Credentials Database 11

Renaming Workspace 12

Deleting a Workspace 13

Verbose Details of Workspaces 13

Exporting Workspace 14

Deleting all Workspaces 14

Importing Hosts 14

Listing Hosts in Database 15

Adding Hosts 16

Deleting a host 16

Exporting Hosts Database 16

Search and Tag Hosts 17

Filter Search 18

Deleting Services of a Host 19

Filter Search 20

Port Specific Services 21

Exporting Services Database 21

Search Services 22

Port Specific Vulnerabilities 22
Service Specific Vulnerabilities 22
Host Specific Vulnerabilities 23
Exporting Vulnerabilities Database 23
Loot Database 24
Search Loot 25
Type Specific Loot 26
Deleting Loot 26
Conclusion 27

Introduction

The database service provided by Metasploit is one of the greatest for keeping a record of your penetration testing activities. 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 of when you enumerated and gathered data so that you can keep track of your activity. To initiate the PostgreSQL database, use the command msfdb init to initialise the database.

msfdb init

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

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

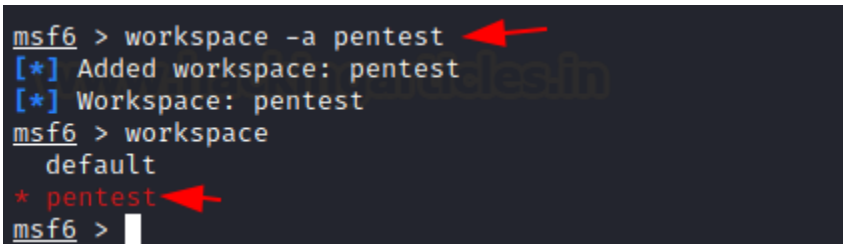
workspace -h

```
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, 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 you are working on as a penetration tester. For the demonstration, we will create a workspace named "pentest." Simply running the workspace without any options will list the various workspaces present in the database. A workspace with an asterisk (*) denotes the current workspace.

workspace -a pentest



Database Nmap Scan

As a part of penetration testing any machine, it is fundamental to perform a port scan. Earlier, we used to perform an Nmap port scan and then export the result into a file for future reference. But with the help of workspace, we can perform an Nmap scan and save the 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 an 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 can see that the options and results that are used and generated are very similar to those that we normally use, with the exception that all of the enumeration that is done is 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.
[*] 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 organisation specifications that require the penetration tester to use Nmap and export the scan results into a predefined format.If that is the case with you, it is still possible to use the workspace database. Let's demonstrate. First, we will be performing a Version Nmap scan on our other targeted hosts at 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 ED
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 microsof
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

msf6 >
```

Services Database

Now we have two hosts, 192.168.1.12 and 192.168.1.16. We have performed db_nmap and Nmap scans 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 use the IP address of your target as a parameter while running the services command.

services
services 192.168.1.16

```
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 8ubunt
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 er
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 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:2
[*] Nmap:             CVE-2001-0554   10.0    https://vulners.com/cve/CVE-2001-0554
[*] Nmap:             PACKETSTORM:105078      7.8     https://vulners.com/packetstor
[*] Nmap:             PACKETSTORM:101052      7.8     https://vulners.com/packetstor
[*] Nmap:             SECURITYVULNS:VULN:8166 7.5     https://vulners.com/securityvu
[*] Nmap:             MSF:ILITIES/OPENBSD-OPENSSH-CVE-2010-4478/      7.5     https:
[*] Nmap:             MSF:ILITIES/LINUXRPM-ELSA-2008-0855/    7.5     https://vulner
[*] 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 in 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

toperform a Bruteforce on the FTP login on our target machine. We can see the correct credentials have made an entry inside the Credentials section.

```
use auxiliary/scanner/ftp/ftp_login
set rhosts 192.168.1.40
set user_file /root/users.txt
set pass_file /root/pass.txt
set stop_on_success true
set verbose false
exploit
creds
```

```
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 Workspace

We previously discussed how to create a workspace, but if you need to rename a workspace for better management, you can do so with the -r option.We first use the workspace command to list the various workspaces. 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 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 a Workspace

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 consiquest 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, documentation is an important part of penetration testing. While working with a workspace, it gets easier to sort your data, but 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 in 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 if you have multiple workspaces and now want to use the nuclear option on all of them, then the -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 work gets deleted or corrupted from the workspace database. We tried to do this, but we were not able to kill the database. We even restarted the system to check, but all the data seemed

toretain 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              2.6.X
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.

Listing Hosts in Database

As we saw earlier, when we perform an Nmap scan with the db_nmap command, we populate the hosts 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 all the 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, 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

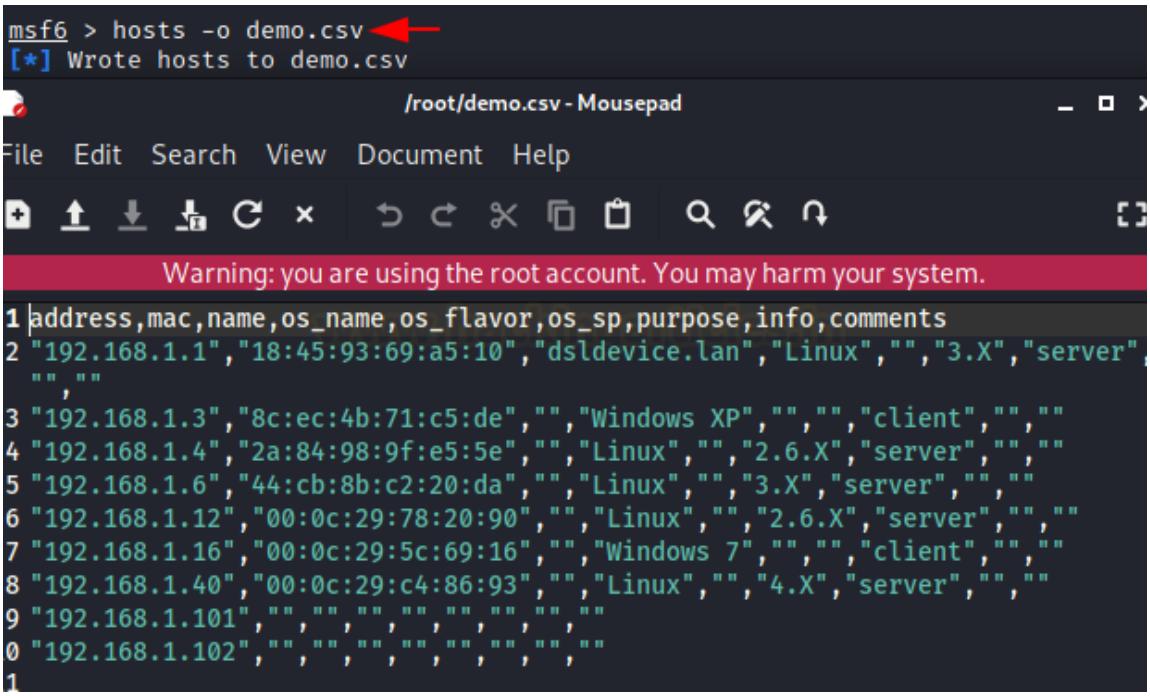
```
msf6 > hosts -d 192.168.1.102

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, 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 if 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
```



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. The Hosts command has the -S option that can help you search for a particular machine based on 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 the 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 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                    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, the hosts table contains a variety of columns. But it is possible to view only the columns that you desire in your host table. It can be done with the help of a filtered search. In the demonstration, we used the -c option to filter only the address, mac, and os_name columns from the hosts table and another example with only the 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     18:45:93:69: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 service 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 the -c option, we can filter our columns such as the host, port, and name of the services as demonstrated below. As host is the primary key in this table, it will be displayed 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 services, the most common way to recognise and differentiate between services is based on their port numbers. As there are default ports for various services, that can 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 a certain port. It is not limited to a single port, we can use multiple ports or 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

So far, we have exported the entire workspace database as well as the 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
```

/root/result.csv - Mousepad

File Edit Search View Document Help

Warning: you are using the root account. You may harm your system.

```
 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 for services based on the hosts, ports, service names, their state, and even the protocol they are running on. Services are running on TCP, UDP, and other protocols. We can search for 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 on the target machine and are enumerated using the Nmap script scan or any of the Metasploit Auxiliary scans. We can search for a specific vulnerability in the service running on the 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-2001-055
                                                                     5/,CVE-2010-4478,SSV:2051
                                                                     /,MSF:ILITIES/ORACLE-SOLA
                                                                     61,CVE-2011-4327,MSF:ILIT
```

Service Specific Vulnerabilities

We just listed the vulnerabilities based on the port number, but as we know, services are not always running on their default ports. With the help of the -s option, we can target the particular service that we are looking for. Here, we presented all the ports from 1 to 65536 and FTP as the service, and we can see the various vulnerabilities for the FTP service no matter which 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 on a particular host. This is one of the most useful representations because it provides a quick overview of the services hosted on the targeted host as well as any potential vulnerabilities.

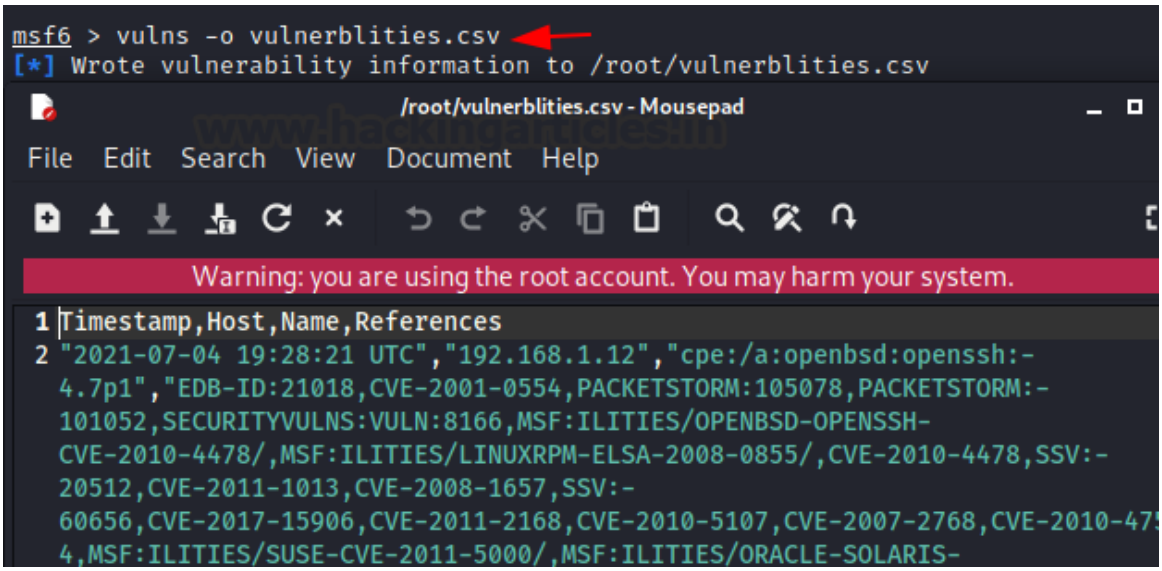
vulns -i 192.168.1.12

```
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/,CV
                                                                         754,MSF:ILITIES/SUSE-C
                                                                         AS-2012-99/,CVE-2012-0
                                                                         259,SECURITYVULNS:VULN
2021-07-04 19:37:17 UTC  192.168.1.12  cpe:/a:isc:bind:9.4.2             SSV:2853,CVE-2008-0122
                                                                         NUXRPM-ELSA-2015-2658/
                                                                         ITPACK:D6DDF5E24DE171D
```

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 host IP address, services running on the machine, and the possible vulnerabilities, it is a preliminary report that can be used during the penetration testing assessments.

vulns -o vulnerblities.csv



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, and Vulnerabilities database, but we saved this one for last. It is called the Loot database. During any penetration testing assessment, there are times where you can exploit a vulnerability and get into the target. This is where you start some post-exploitation activities, including enumerating for credentials and hashes. If you can extract those using the Metasploit Post Exploitation Module, they 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 configuration files that might contain some passwords or important information stored in them.

```
use post/linux/gather/enum_configs
set session 1
exploit
```

```
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, the enum_configs grabbed a bunch of config files. 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 moment. We can use the search option to search for a specific loot, much 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/20210704154713_
```

Type Specific Loot

As it is possible to be working on multiple targets, it is also possible to gather different types of loot 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 the -t option as demonstrated below.

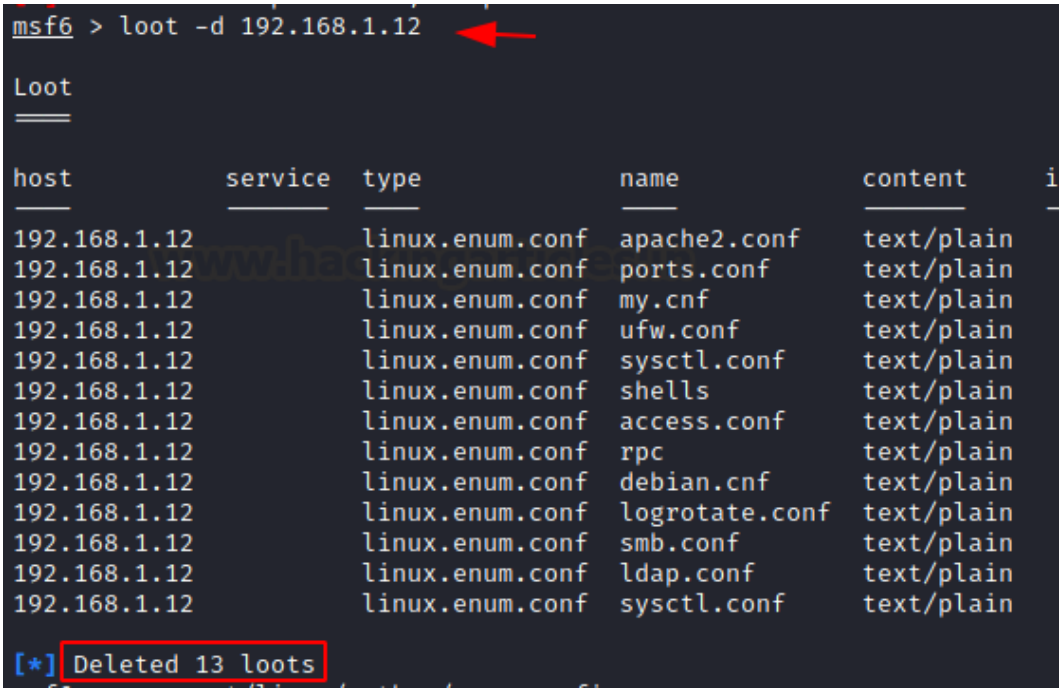
loot -t linux.enum.conf

```
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 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, if we want to delete the loot of a specific target, we can provide the IP address of that target and delete all the loot from that source.

loot -d 192.168.1.12



Conclusion

This was a learning experience, as when we start with the penetration activities, we tend not to focus on the documentation process or provide your work with a proper structure and backup. However, with time and a few instances where a lack of these qualities proves to be advantageous. The workspace and the database function of Metasploit are not new features; they have been around for years, and yet the general usage of these in real-life seems very low. Hence, it inspired us to provide the guide, so that lots of penetration testers can use it and benefit from it.