ARYAMAN MISHRA 19BCE1027

LAB 6

## EX-6-METASPLOIT INFORMATION GATHERING USING NMAP

**PREPARING METASPLOIT FOR PORT SCANNING**

Scanners and most other auxiliary modules use the ‘RHOSTS’ option instead of ‘RHOST’. RHOSTS can take IP ranges (192.168.1.20-192.168.1.30), CIDR ranges (192.168.1.0/24), multiple ranges separated by commas (192.168.1.0/24, 192.168.3.0/24), and line-separated host list files (file:/tmp/hostlist.txt). This is another use for a grepable Nmap output file.

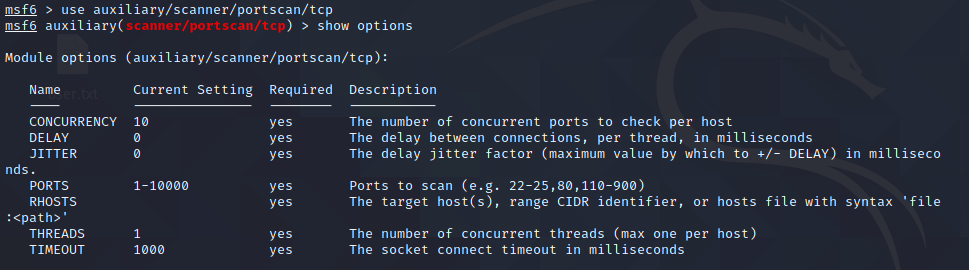
By default, all of the scanner modules will have the ‘THREADS’ value set to ‘1’. The ‘THREADS’ value sets the number of concurrent threads to use while scanning. Set this value to a higher number in order to speed up your scans or keep it lower in order to reduce network traffic but be sure to adhere to the following guidelines:

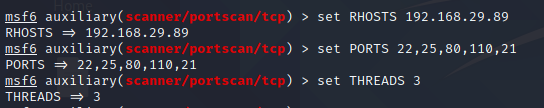
* Keep the THREADS value under 16 on native Win32 systems
* Keep THREADS under 200 when running MSF under Cygwin
* On Unix-like operating systems, THREADS can be set as high as 256.

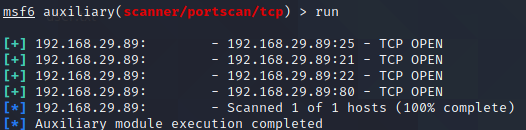
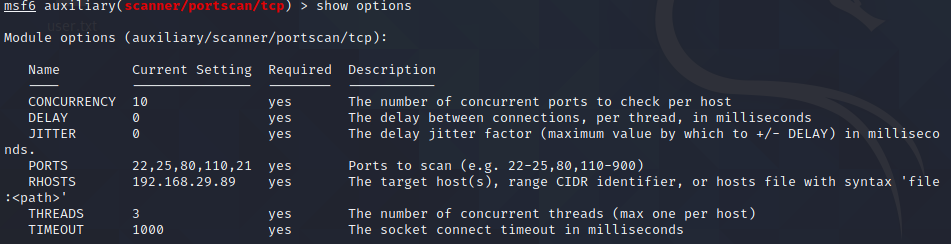
## NMAP & DB\_NMAP

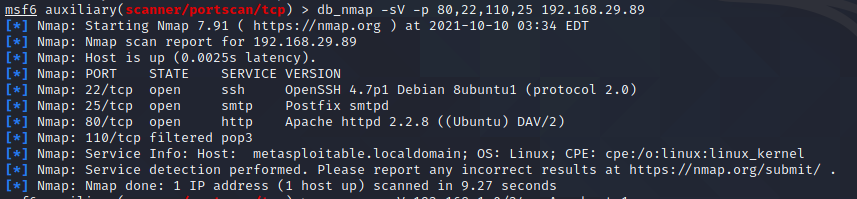
We can use the **db\_nmap** command to run [Nmap](http://tools.kali.org/information-gathering/nmap) against our targets and our scan results would than be stored automatically in our database. However, if you also wish to import the scan results into another application or framework later on, you will likely want to export the scan results in XML format. It is always nice to have all three Nmap outputs (xml, grepable, and normal). So we can run the Nmap scan using the **-oA** flag followed by the desired filename to generate the three output files, then issue the **db\_import** command to populate the Metasploit database.

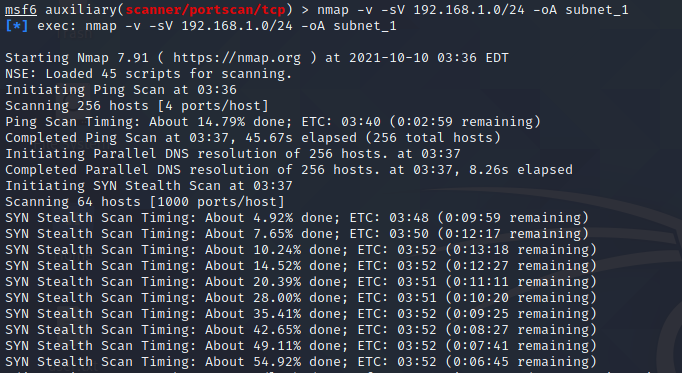
Run Nmap with the options you would normally use from the command line. If we wished for our scan to be saved to our database, we would omit the output flag and use **db\_nmap**. The example below would then be **db\_nmap -v -sV 192.168.1.0/24**.



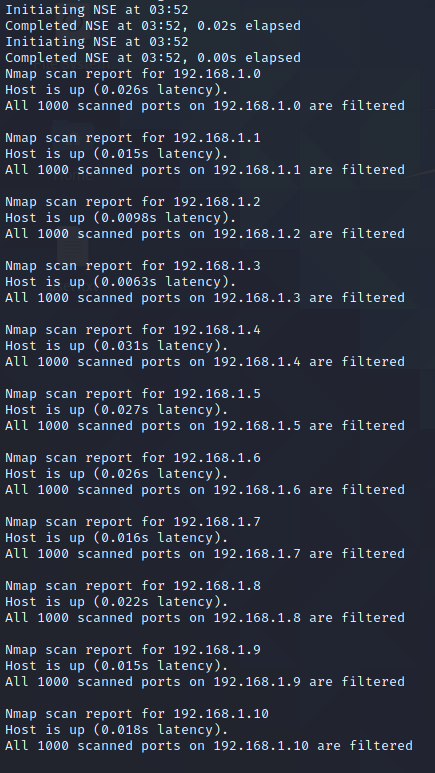




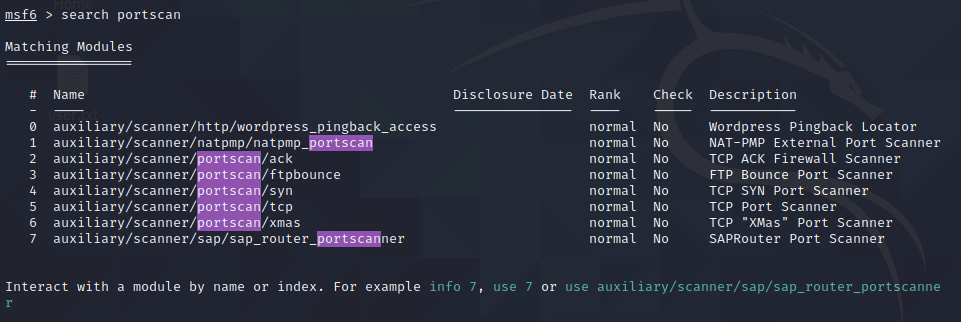


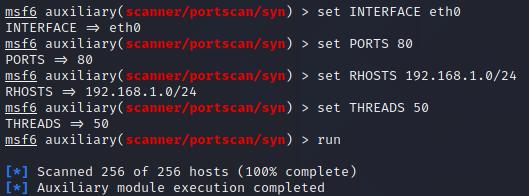
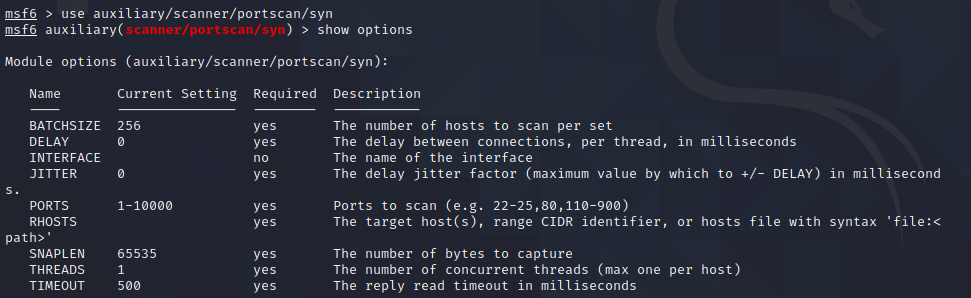


For the sake of comparison, we’ll compare our Nmap scan results for port 80 with a Metasploit scanning module. First, let’s determine what hosts had port 80 open according to Nmap.



## PORT SCANNING

In addition to running Nmap, there are a variety of other port scanners that are available to us within the framework. 



For the sake of comparison, we’ll compare our Nmap scan results for port 80 with a Metasploit scanning module. First, let’s determine what hosts had port 80 open according to Nmap.

msf > cat subnet\_1.gnmap | grep 80/open | awk '{print $2}'

[\*] exec: cat subnet\_1.gnmap | grep 80/open | awk '{print $2}'

192.168.1.1

192.168.1.2

192.168.1.10

192.168.1.109

192.168.1.116

192.168.1.150

The Nmap scan we ran earlier was a [SYN scan](https://nmap.org/book/man-port-scanning-techniques.html) so we’ll run the same scan across the subnet looking for port 80 through our eth0 interface, using Metasploit.

msf > use auxiliary/scanner/portscan/syn

msf auxiliary(syn) > show options

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

Name Current Setting Required Description

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

BATCHSIZE 256 yes The number of hosts to scan per set

DELAY 0 yes The delay between connections, per thread, in milliseconds

INTERFACE no The name of the interface

JITTER 0 yes The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.

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 500 yes The reply read timeout in milliseconds

msf auxiliary(syn) > set INTERFACE eth0

INTERFACE => eth0

msf auxiliary(syn) > set PORTS 80

PORTS => 80

msf auxiliary(syn) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(syn) > set THREADS 50

THREADS => 50

msf auxiliary(syn) > run

[\*] TCP OPEN 192.168.1.1:80

[\*] TCP OPEN 192.168.1.2:80

[\*] TCP OPEN 192.168.1.10:80

[\*] TCP OPEN 192.168.1.109:80

[\*] TCP OPEN 192.168.1.116:80

[\*] TCP OPEN 192.168.1.150:80

[\*] Scanned 256 of 256 hosts (100% complete)

[\*] Auxiliary module execution completed

Here we’ll load up the ‘tcp’ scanner and we’ll use it against another target. As with all the previously mentioned plugins, this uses the ‘RHOSTS’ option. Remember we can issue the **hosts -R** command to automatically set this option with the hosts found in our database.

msf > use auxiliary/scanner/portscan/tcp

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

DELAY 0 yes The delay between connections, per thread, in milliseconds

JITTER 0 yes The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.

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

RHOSTS yes The target address range or CIDR identifier

THREADS 1 yes The number of concurrent threads

TIMEOUT 1000 yes The socket connect timeout in milliseconds

msf auxiliary(tcp) > hosts -R

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

RHOSTS => 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 1-1024 yes Ports to scan (e.g. 22-25,80,110-900)

RHOSTS 172.16.194.172 yes The target address range or CIDR identifier

SNAPLEN 65535 yes The number of bytes to capture

THREADS 10 yes The number of concurrent threads

TIMEOUT 1000 yes The socket connect timeout in milliseconds

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

[\*] 172.16.194.172:111 - TCP OPEN

[\*] 172.16.194.172:139 - TCP OPEN

[\*] 172.16.194.172:445 - TCP OPEN

[\*] 172.16.194.172:514 - TCP OPEN

[\*] 172.16.194.172:513 - TCP OPEN

[\*] 172.16.194.172:512 - TCP OPEN

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

[\*] Auxiliary module execution completed

msf auxiliary(tcp) >

We can see that Metasploit’s built-in scanner modules are more than capable of finding systems and open ports for us. It’s just another excellent tool to have in your arsenal if you happen to be running Metasploit on a system without Nmap installed.

SMB VERSION SCANNING

Now that we have determined which hosts are available on the network, we can attempt to determine the operating systems they are running. This will help us narrow down our attacks to target a specific system and will stop us from wasting time on those that aren’t vulnerable to a particular exploit.

Since there are many systems in our scan that have port 445 open, we will use the **scanner/smb/version** module to determine which version of Windows is running on a target and which [Samba](https://www.samba.org/) version is on a Linux host.

msf > use auxiliary/scanner/smb/smb\_version

msf auxiliary(smb\_version) > set RHOSTS 192.168.1.200-210

RHOSTS => 192.168.1.200-210

msf auxiliary(smb\_version) > set THREADS 11

THREADS => 11

msf auxiliary(smb\_version) > run

[\*] 192.168.1.209:445 is running Windows 2003 R2 Service Pack 2 (language: Unknown) (name:XEN-2K3-FUZZ) (domain:WORKGROUP)

[\*] 192.168.1.201:445 is running Windows XP Service Pack 3 (language: English) (name:V-XP-EXPLOIT) (domain:WORKGROUP)

[\*] 192.168.1.202:445 is running Windows XP Service Pack 3 (language: English) (name:V-XP-DEBUG) (domain:WORKGROUP)

[\*] Scanned 04 of 11 hosts (036% complete)

[\*] Scanned 09 of 11 hosts (081% complete)

[\*] Scanned 11 of 11 hosts (100% complete)

[\*] Auxiliary module execution completed

Also notice that if we issue the **hosts** command now, the newly-acquired information is stored in Metasploit’s database.

msf auxiliary(smb\_version) > hosts

Hosts

=====

address mac name os\_name os\_flavor os\_sp purpose info comments

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

192.168.1.201 Microsoft Windows XP SP3 client

192.168.1.202 Microsoft Windows XP SP3 client

192.168.1.209 Microsoft Windows 2003 R2 SP2 server

**IDLE SCANNING**

Nmap’s IPID Idle scanning allows us to be a little stealthy scanning a target while spoofing the IP address of another host on the network. In order for this type of scan to work, we will need to locate a host that is idle on the network and uses IPID sequences of either Incremental or Broken Little-Endian Incremental. Metasploit contains the module **scanner/ip/ipidseq** to scan and look for a host that fits the requirements.

In the free online Nmap book, you can find out more information on [Nmap Idle Scanning](http://nmap.org/book/idlescan.html).

msf > use auxiliary/scanner/ip/ipidseq

msf auxiliary(ipidseq) > show options

Module options (auxiliary/scanner/ip/ipidseq):

Name Current Setting Required Description

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

INTERFACE no The name of the interface

RHOSTS yes The target address range or CIDR identifier

RPORT 80 yes The target port

SNAPLEN 65535 yes The number of bytes to capture

THREADS 1 yes The number of concurrent threads

TIMEOUT 500 yes The reply read timeout in milliseconds

msf auxiliary(ipidseq) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(ipidseq) > set THREADS 50

THREADS => 50

msf auxiliary(ipidseq) > run

[\*] 192.168.1.1's IPID sequence class: All zeros

[\*] 192.168.1.2's IPID sequence class: Incremental!

[\*] 192.168.1.10's IPID sequence class: Incremental!

[\*] 192.168.1.104's IPID sequence class: Randomized

[\*] 192.168.1.109's IPID sequence class: Incremental!

[\*] 192.168.1.111's IPID sequence class: Incremental!

[\*] 192.168.1.114's IPID sequence class: Incremental!

[\*] 192.168.1.116's IPID sequence class: All zeros

[\*] 192.168.1.124's IPID sequence class: Incremental!

[\*] 192.168.1.123's IPID sequence class: Incremental!

[\*] 192.168.1.137's IPID sequence class: All zeros

[\*] 192.168.1.150's IPID sequence class: All zeros

[\*] 192.168.1.151's IPID sequence class: Incremental!

[\*] Auxiliary module execution completed

Judging by the results of our scan, we have a number of potential zombies we can use to perform idle scanning. We’ll try scanning a host using the zombie at 192.168.1.109 and see if we get the same results we had earlier.

msf auxiliary(ipidseq) > nmap -Pn -sI 192.168.1.109 192.168.1.114

[\*] exec: nmap -Pn -sI 192.168.1.109 192.168.1.114

Starting Nmap 5.00 ( http://nmap.org ) at 2009-08-14 05:51 MDT

Idle scan using zombie 192.168.1.109 (192.168.1.109:80); Class: Incremental

Interesting ports on 192.168.1.114:

Not shown: 996 closed|filtered ports

PORT STATE SERVICE

135/tcp open msrpc

139/tcp open netbios-ssn

445/tcp open microsoft-ds

3389/tcp open ms-term-serv

MAC Address: 00:0C:29:41:F2:E8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 5.56 seconds