Banner Grabbing Using Telnet

The tried-and-true manual technique for enumerating banners and application information has traditionally been based on Telnet.

In this exercise, you will open a Telnet connection to various TCP ports on the target system and record the banner information that is presented.

Exercise 1: Banner grabbing using Telnet:

1. From your BackTrack system, open a shell and type the following (only type what's in **bold**):

user1@bt:~#**telnet target_IP_address 80** (hit RETURN a few times) – you may also need to hit ESC a few times

Syntax breakdown:

telnet: program name

target_IP_address: the IP address of the target system

80: open TCP port on target system

2. What Web server application is running on the target system?

3. Repeat step #1, replacing the TCP port number with those ports identified in previous labs.

4. Record your results here:

PORT #	*BANNER INFORMATION*
----------	----------

Banner Grabbing Using Netcat

Netcat is a networking utility that reads and writes data across network connections, using the TCP/IP protocol. It is designed to be a reliable "back-end" tool that can be used directly, or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool (since it can create almost any kind of connection you would need and has several interesting built-in capabilities).

It provides access to the following main features:

- Outbound or inbound connections, TCP or UDP, to or from any ports
- Full DNS forward/reverse checking, with appropriate warnings
- Ability to use any local source port
- Ability to use any locally-configured network source address
- Built-in port-scanning capabilities, with randomizer
- Built-in loose source-routing capability
- Can read command line arguments from standard input
- Optional ability to let another program service inbound connections

In the following exercises, you will use Netcat to grab application banners using various techniques.

Exercise 1: Banner grabbing using Netcat:

5. From a BackTrack shell, type the following (only type what's in **bold**):
user1@bt:~# **nc -v target_IP_address 80** (wait a few seconds for Netcat to try and resolve the target hostname into an IP address)

Syntax breakdown:

nc: program name

-v: verbose mode

target_IP_address: the IP address of the target system

80: open TCP port on target system

6. Then (only type what's in **bold**):

GET /index.html HTTP/1.0 - hit RETURN a few times - (NOTE: you can also use 1.1 instead of 1.0)

7. List the Web server software vendor and version running on port 80 on the target system:

8. Repeat step #1, using various open TCP port numbers, as identified in previous labs

Exercise 2: Banner grabbing using Netcat with a nudge file: in this exercise, you will direct the contents of a file into Netcat to *nudge* remote systems for even more information:

1. Using nano, create a file called nudge:
user1@bt:~# **nano nudge**

2. In the nudge file, type the following line (only type what's in **bold**):

HEAD / HTTP/1.0, followed by 2 carriage returns

The HTTP HEAD command asks the target system to not return the requested page, but rather, just the information about the page. It's usually used by a browser or proxy server to see if it can use a cached version of a page rather than pull the whole page, amongst other things.

3. Save the nudge file and close nano

4. At the prompt, type the following (only type what's in **bold**):
user1@bt:~# **nc -nvv -o /root/ceh/banners target_IP_address 80 < /root/ceh/nudge**

Syntax breakdown:

nc: program name

-nvv: program option to use numeric addresses only and extra verbose mode

-o /root/ceh/banners: program option to dump the target system's banner information in HEX format to a file called banners in the /root/ceh directory

target_IP_address: the IP address of the target system

80: open TCP port on target system

< /root/ceh/nudge: read into Netcat what's in the /root/ceh/nudge file

5. Once the command completes, open the /root/ceh/banners file. List the software vendor and version running on port 80 on the target system:

Active Stack Fingerprinting Using nmap

Nmap employs the various active stack fingerprinting techniques mentioned in your slides (except for the fragmentation handling and ICMP error message queuing) by using the -O option.

Remember, the accuracy of the determination of the remote OS is largely dependent on at least one open port on the target system.

In the next few exercises, you will be using nmap on a Linux machine to identify the OS running on the target system(s).

Exercise 1: Active stack fingerprinting option:

9. From a BackTrack shell, type the following (only type what's in **bold**):
user1@bt:~# **nmap -O -v target_IP_address > /root/ceh/os_detect**

Syntax breakdown:

nmap: program name

-O: this is the letter O - not the number zero; program option for active stack fingerprinting of remote OS

-v: **verbose mode**

target_IP_address: IP address of the target system

> /root/ceh/os_detect: redirect the output to a file called os_detect in the /root/ceh directory

10. Examine your results:

    user1@bt:~#**cat /root/ceh/os_detect | less**

11. What OS did nmap determine is running on the target system? Is it correct?

Exercise 2: Putting it all together: in this exercise, you will combine TCP

and UDP port scans with application version detection and OS fingerprinting,

in one command:

1. The syntax to perform this is (only type what's in **bold**, on one line):

user1@bt:~# **nmap -A -sS -sU -v target_IP_address > /root/ceh/os_detect1**

Syntax breakdown:

nmap: program name

-A: program option combining application vendor/version detection on open ports and active stack fingerprinting of remote OS

-sS: program option for TCP SYN scan

-sU: program option for UDP port scan

-v: **verbose mode**

target_IP_address: the IP address of the target system

> /root/ceh/os_detect1: redirect the output to a file called os_detect1 in the /root/ceh directory

2. Examine your results:
   user1@bt:~#**cat /root/ceh/os_detect1 | less**

3. What OS did nmap determine is running on the target system? Is it correct?

4. Record the application vendor and version running on the various
UDP and TCP ports:

5. Repeat steps #1-3 using a different target IP address. Did nmap correctly identify the target system's OS?

Passive Stack Fingerprinting Using Ettercap

Exercise 1: Ettercap and HTTP:

1. In this exercise, you will start Ettercap, connect to your target Web site(s), and passively watch the connection to determine the target system's OS.

2. From a BackTrack shell, type the following (only type what's in **bold**):

user1@bt:~# **ettercap -C**

Syntax breakdown:

ettercap: program name

-C: Ncurses-based GUI

3. Select Sniff/Unified Sniffing - make sure eth0 is listed as the interface to sniff traffic on. Hit RETURN

4. Select Start/Start Sniffing

5. Select View/Profiles

6. Open a Web browser

7. Go to the target Web site (use target's IP address)

8. Select the target's IP in the Ettercap Profiles window. Hit RETURN to see OS information

9. To get back to the Ettercap Profiles window, hit TAB 2 times and then RETURN. Arrow down to Profiles. Hit RETURN

10. Repeat step #6, selecting a different target Web site

11. How successful was Ettercap in fingerprinting the target OS(es)?

FTP Enumeration

Hydra is a parallelized login cracker that supports numerous protocols. It can perform rapid dictionary attacks against more than 30 protocols, including Telnet, FTP, HTTP, SMB, several databases, and more. **Make sure that FTP in your windows 2003 is open for connection.**

Exercise 1: brute-forcing FTP accounts: in this exercise, you will use Hydra w/a

dictionary file to guess the password for a given username:

1. From a BackTrack shell, type the following (only type what's in **bold**):

   user1@bt:~#**cd /pentest/passwords/wordlists**

   user1@bt:~#**pwd**

   Syntax breakdown:

   cd /pentest/passwords/wordlists: change into the directory /pentest/passwords/wordlists

   pwd: program name to print current directory

2. Next, create a simple dictionary file with a few passwords to test Hydra by typing the following (only type what's in **bold**, on one line):

   user1@bt:~#**echo passwd > hydra.lst**

   user1@bt:~#**echo password >> hydra.lst**

   Syntax breakdown:

   echo passwd > hydra.lst: re-direct the word *passwd* into a file called *hydra.lst* located in the */pentest/passwords/wordlists* directory

   echo password >> hydra.lst: re-direct the word *password* and append it into the file called *hydra.lst* located in the */pentest/passwords/wordlists* directory

3. Then type (only type what's in **bold**):
   user1@bt:~#**hydra -l administrator -P hydra.lst -v -t18 win_target_IP_address ftp**

   Syntax breakdown:

   hydra: program name

   -l administrator: program option to login with username *administrator*

-P hydra.lst: program option to load passwords from file called *hydra.lst* (located in BackTrack VM: /pentest/passwords/wordlists)

-v: verbose mode

-t18: program option to run 18 connection attempts in parallel

win_target_IP_address: the IP address of the Windows XP target system

ftp: program option for the service to crack (FTP in this exercise)

NOTE: if you want to use a larger password file, replace **hydra.lst** with **darkc0de.lst**

4. Did Hydra find the password for the administrator's account? If so, what is it?

SSH Enumeration

BruteSSH is a simple SSH password brute forcing utility. **Make sure that SSH port your target BT is open for connection.**

Exercise 1: brute-forcing an SSH user account: in this exercise, you will use the BruteSSH

Python script in BackTrack to crack the password of a given user account:

1. From a BackTrack shell, type the following (only type what's in **bold**):

   user1@bt:~#**cd /pentest/passwords/brutessh**

   user1@bt:~#**pwd**

   Syntax breakdown:

   cd /pentest/passwords/brutessh: change into the directory /pentest/passwords/brutessh

   pwd: program name to print current directory

2. Create a simple dictionary file with a few passwords in it to test the BruteSSH script by typing the following (only type what's in **bold**, on one line):

   user1@bt:~#**echo passwd > /pentest/passwords/wordlists/ssh.lst**

   user1@bt:~#**echo password >> /pentest/passwords/wordlists/ssh.lst**

   Syntax breakdown:

   echo passwd > /pentest/passwords/wordlists/ssh.lst: re-direct the word *passwd* into a file called *ssh.lst* located in the */pentest/passwords/wordlists* directory

   echo password >> /pentest/passwords/wordlists/ssh.lst: re-direct the word *password* and append it into the file called *ssh.lst* located in the */pentest/passwords/wordlists* directory

3. Then run the BruteSSH Python script (only type what's in **bold**, on one line):

   user1@bt:~#**./brutessh.py -h Linux_target_IP_address -u root -d /pentest/passwords/wordlists/ssh.lst**

   Syntax breakdown:

   ./brutessh.py: Python script to run

   -h Linux_target_IP_address: the IP address of the Linux target system

-u root: program option for user account to login with

-d /pentest/passwords/wordlists/ssh.lst: program option for password file to use

NOTE: if you want to use a larger password file, replace
/pentest/passwords/wordlists/ssh.lst with
/pentest/passwords/wordlists/darkc0de.lst

SMTP Enumeration Using Telnet

SMTP provides two built-in commands that allow for the enumeration of users:

- **VRFY**: confirms names of valid users, and
- **EXPN**: reveals the actual delivery addresses of aliases and mailing lists

Exercise 1: SMTP enumeration using Telnet: in this exercise, you will use Telnet to connect to an SMTP server on TCP port 25 and issue the VRFY and EXPN commands to further enumerate the target system:

1. From a BackTrack shell, type the following (only type what's in **bold**):
   user1@bt:~# **telnet target_IP_address 25**

2. When you see the 220 message/application banner hit RETURN

3. Type **vrfy root** (hit RETURN)

4. Is root a valid username on the target system? How can you tell?

5. Type **expn ptest** (hit RETURN)

6. Is ptest a valid mail address on the target system? How can you tell?

7. Repeat steps #3 and 5, trying various usernames and mail addresses. Record any successes here:

HTTP Enumeration Using Nikto

Nikto is an open source web server scanner that performs comprehensive tests against web servers for multiple items, including thousands of potentially dangerous files and vulnerable web server software versions.

Exercise 1: web server scanning using Nikto: in this exercise, you will use Nikto from

BackTrack to scan multiple web servers:

1. From a BackTrack shell, type the following (only type what's in **bold**):

   user1@bt:~#**cd /pentest/scanners/nikto**

   user1@bt:~#**pwd**

   Syntax breakdown:

   cd /pentest/passwords/nikto: change into the directory /pentest/passwords/nikto

   pwd: program name to print current directory

2. Update the Nikto databases and plugins from cirt.net by typing the following (only type what's in **bold**):

   user1@bt:~#**./nikto.pl -update**

   Syntax breakdown:

   ./nikto.pl: PERL script to run

   -update: program option to update Nikto databases and plugins

3. Run the Nikto PERL script to scan a Windows target web server (only type what's in **bold**, on one line):

   user1@bt:~#**./nikto.pl -h win_target_IP_address > /root/ceh/nikto_win_scan**

   Syntax breakdown:

   ./nikto.pl: PERL script to run

   -h win_target_IP_address: the IP address of the Windows target system

   > /root/ceh/nikto_win_scan: redirect the output to a file called nikto_win_scan in the /root/ceh directory

4. Examine your results:

   user1@bt:~#**cat /root/ceh/nikto_win_scan | less**

5. Record your results:

6. Repeat step #3 using your UNIX target IP address (only type what's in **bold**, on one line):

   user1@bt:~#**./nikto.pl -h unix_target_IP_address > /root/ceh/nikto_unix_scan**

7. Examine your results:

   user1@bt:~#**cat /root/ceh/nikto_unix_scan | less**

8. Record your results: