Network Tools

Saurabh Barjatiya

2012-03-19 Mon

Contents

Note: Commands and configuration files mentioned below are with respect to Cent-OS / Fedora / Red-hat based systems. Same content should apply to other distributions with minor changes in command options, location of configuration files, etc.

1 Port scanning (`nmap`)

It is always good to verify after proper firewall (such as `iptables`) configuration has been done that only desired services are accessible from external users. An easy way to accomplish this is to port scan protected server from prospective client machines / potential attackers to check which ports are open. One very versatile port scanning tool available for Linux (also works equally well with Windows) is `nmap`. `nmap` is well documented and we can use:

```
nmap
```

command without arguments to see various options supported by nmap and try each of them. `man nmap` can be used to get detailed information on each of the options available.

Some important nmap command line options are explained below:

Option	Description
-sP	Ping scan. Only determine whether host is online or not. Do not scan ports. This is very useful to find out which machines are on in given IP range.
-p	Scan only these ports. Useful to check for a particular service on given range of IPs
-n	Do not do DNS resolution. Very useful for passive information gathering.
-sS	TCP SYN Scan
-sA	TCP ACK Scan
-sT	TCP Connect Scan
-sU	UDP Scan
-sV	Probe open ports to determine service/version information
-O	Enable OS detection
-f [–mtu]	Fragment packets optionally with given MTU
-D	Use given set of decoys before scanning target with our address
-S <IP>	Use spoofed source address
-e	Use specified interface
–spoof-mac	Spoof MAC address
–ttl	Send packets with given TTL
–badsum	Send packets with bad checksum
-v	Increase verbosity. Can be used twice for greater effect
–open	Only show open (or possibly open) ports
-6	Enable IPv6 scanning
-A	Enable OS detection, service detection and traceroute

2 Capturing packets at command line (`tcpdump`)

2.1 About `tcpdump`

`tcpdump` is a very powerful command-line based packet capturing tool. It can be used to display information about captured packets on screen or capture packets in pcap file format for later analysis. tcpdump has very small footprint and can be used to capture packets even when there is heavy network I/O. 'tcpdump' accepts filters in kernel filter format so that only the packets we are interested in get captured very efficiently. tcpdump uses `libpcap` for its packet capture operations.

2.2 Useful command line options

Some very useful command line options supported by tcpdump are:

Option	Description
-c	Stop capturing after count number of packets have been captured
-C	Do not store more then this many MBs (1,000,000 bytes) in single capture file. When file size exceeds it starts new file with suffixes .1, .2 etc. added between consecutive files.
-D	List interfaces available for capturing along with description.
-i	interface name or number (as shown by -D) on which packets should be captured. Note by default it will choose lowest numbered interface which is up and not a loopback device.
-n	Do not convert IP address to hostnames via reverse DNS lookup. This option is very important if we are capturing packets on heavy traffic links to avoid too many DNS lookups which may affect packet capture or generate significant DNS traffic.
-nn	If performance is issue or port numbers are preferred over service names then we can use -nn to avoid converting of port numbers to service names, like 22 to ssh or 80 to http. This does not generates any additional traffic as mostly file /etc/services would be used to convert port numbers to service names, but can require some small processing.
-p	Do not put interface in promiscuous mode. Note that interface can already be in promiscuous mode and in that case tcpdump would end up capturing packets meant for other hosts in hub like networks.
-r <file>	Read packets from given file and not from live interface.
-s <size>	Capture only first specified number of bytes of each packets. This is useful if we are interested only in protocol (TCP/IP/UDP, etc.) headers and not in application payload. To capture entire packet the size or snaplen can be specified as '0'
-v	Generate verbose output. We can use -vv or -vvv to increase verbosity level
-q	Generate quieter (lesser) output
-w <file>	Write output to file.
-A	Print information contained in packets in ASCII format
-x	Print information contained in packets in hexadecimal format.

Note:

- A pseudo-interface named all is also shown among with other interfaces. But capturing on any interface has limitation that it can be done only in non-promiscuous mode. We cant capture packets on any interface in promiscuous mode.
- If we want to capture packets only meant for current host then we can use filter 'ether host <host-mac-address> or ether broadcast'. This would work even if interface is in promiscuous mode.
- We can specify filename as '-' to -r or -w options so that input is taken from stdio or output is written to stdout.

2.3 `tcpdump` filter format

We can specify filters (conditions which must be satisfied) for the packets to be captured. Various filter options are:

Expression	Meaning
host <ip-address>	Only packets to or from the specified IP address are captured
src host <ip-address>	Only packets with matching source IP address are captured
dst host <ip-address>	Only packets with matching destination IP address are captured
port <number>	Only packets with source/destination TCP/UDP port specified as argument are captured.
src port <number>	Only packets with source TCP/UDP port specified as argument are captured.
dst port <number>	Only packets with destination TCP/UDP port specified as argument are captured.
<protocol>	Only packets of mentioned protocol will be captured. Accepted protocol names are ip, arp, rarp, tcp, udp, wlan, ip6 and ether.
and, or, not	We can combine multiple expressions with and, or, not
ether host <mac-address>	Allow only with matching source or destination mac address.
ether src <mac-address>	Capture packets only with specified source mac address
ether dst <mac-address>	Capture packets only with specified destination mac address
gateway <host>	Packet was sent or received via host as gateway. Note for this the information about host's MAC address must be present in /etc/ethers' file. Also host must be either resolvable by DNS or its IP information should be mentioned in '/etc/hosts' file.
net <network-number>	Captures packets only when source/destination IP belongs to given network.
net <net> mask <netmask>	Packet matches network with specified netmask specified in dotted decimal format
net <net>/<len>	Packet matches network with specified netmask using bit-mask length notation.
portrange <port1>-<port2>	Port number lies within given range.
src portrange <port1>-<port2>	Source port lies within given range
dst portrange <port1>-<port2>	Capture if destination port lies within given range
less <length>	Capture if packet length is less then specified length

greater <length>	Capture if packet length is greater then specified length
ether broadcast	Capture if ethernet broadcast packet
ip broadcast	Capture if IP broadcast packet
ether multicast	Capture if ethernet multicast packet
ip multicast	Capture if IP multicast packet
vlan <vlan-id>	Capture if the packet is an IEEE 802.1Q VLAN packet. If <vlan_{id}> is specified, only true if the packet has the specified vlan_{id}. Note that the first vlan keyword encountered in expression changes the decoding offsets for the remainder of expression on the assumption that the packet is a VLAN packet. The vlan [vlan_{id}] expression may be used more than once, to filter on VLAN hierarchies. Each use of that expression increments the filter offsets by 4. Read man page to understand this option properly

Note:

- We can combine protocol (ip, tcp, etc.), direction (src, dst) and port in single expressions like 'tcp dst port 80'
- There is also very powerful indexing operation to access byte at specific location in packet which then can be compared using <, <, >=, <=, =, !=, &, | etc. C language operators with other byte or decimal constant. Complete information on this can be found in man page.

Additional information (including above mentioned information) can be found at tcpdump man page

3 Capturing and analyzing packets (`wireshark`)

`wireshark` is packet capturing and analysis tool with GUI interface. Wireshark is very powerful network forensic tool which understands many diverse protocols which are used over Internet. Wireshark also provides statistics, Protocol specific details, display filters, option for TCP reassembly, etc. making it very useful and unique.

One needs administrator or root privileges to run `wireshark`. It is very intuitive to use and one can learn considerable usage of `wireshark` by playing around with it for few hours.

4 Host based IPS / IDS (`snort`)

`snort` is a very powerful host based IPS / IDS with very powerful and flexible rule syntax. It has many different configuration options and we can get good initial IPS ruleset to be used with snort from web.

4.1 Snort modes

Snort can be used in three different modes:

- Sniffing - `-v` - In this mode snort just sniffs the packets and displays relevant information on screen.
- Logging - `-l` - In this mode snort logs packets in log files. We can use these log files for analysis later on. We can also use '-b' option to log in binary format (libpcap) format which can be used by wireshark / tcpdump etc.
- Network Intrusion Detection System (NIDS) - `-c` {snort-configuration-file} - In this mode snort uses set of rules and inspects packets for matching rules and takes action as specified in the rules.

We can combine more than one mode together to do NIDS, logging, etc. together.

4.2 Configuring snort rules

Following steps can be used for configuring or testing very basic snort rules:

- Create file /etc/snort/rules/local.rules using 'touch /etc/snort/rules/local.rules'
- In file '/etc/snort/snort.conf' uncomment line 'include \$RULE$_{\text{PATH}}$/local.rules'
- Now we can put simple rules in local.rules file and test them with snort.

4.2.1 Snort rule syntax

Snort rules are in format

```
action   protocol   src_ip  src_port  direction   dst_ip   dst_port   (rule options)
```

Note:

- Most snort rules are written in single line. We can write rules that span multiple lines by ending all but-last line with a backslash ('́) character.

- It is not necessary for rules to have rule options, but most rules would have options to make them useful.

Sample rule is:

```
alert tcp any any <> 10.100.1.107 80 (flags:S; msg: "HTTP access on vm7"; sid:1000001;
```

Here types of action can be alert, log, pass, activate, drop, reject, sdrop etc. Protocol can be IP, ICMP, TCP, UDP, etc. Source or Destination can be IP, IP/mask, Groups of IP/Mask in square brackets, Negation of IP, Negation of groups of IP etc. Port can be single port, range of port or group of ports. Direction can be -> or <> to indicate unidirectional or bi-directional connection. Snort has lot of rule options. One can refer to snort users manual hosted at http://www.snort.org/docs to get documentation / examples on various options available.

4.2.2 Example snort rules

Some example rules are:

```
alert tcp any any -> any 80 (content:"BOB"; gid:1000001; sid:1; rev:1;)
alert tcp any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
        content:"expn root"; nocase; classtype:attempted-recon;)
alert tcp any any -> any 80 (msg:"WEB-MISC phf attempt"; flags:A+; \
        content:"/cgi-bin/phf"; priority:10;)
alert tcp any any -> any 80 (msg:"EXPLOIT ntpdx overflow"; \
        dsize:>128; classtype:attempted-admin; priority:10 );
alert tcp any any -> any any (content:"ABC"; content:"DEF"; distance:1;)
alert tcp any any -> any any (content:"ABC"; content:"EFG"; within:10;)
alert tcp any any -> any 80 (content:"ABC"; content:"EFG"; http_client_body;)
```

4.2.3 Installing snort

Installing snort on stable distributions like Cent-OS which have very old and stable versions of most libraries is not easy as snort developers use many latest libraries in snort development. Hence for snort beginners it is best to try snort on latest distribution of Fedora or Ubuntu on which pre-built binaries of latest snort version can be installed and used without any problem.

5 Password cracking using bruteforce, dictionary attacks (`john`)

`john` is very advanced password cracking software which can perform sophisticated bruteforce attacks / dictionary based attacks on given set of password hashes. Multiple john sessions can be run on same system, each of which can be paused / resumed as and when required. John can be used to crack both Windows (NTLM) or Linux (md5crypt) based hashes. To crack Windows passwords john can be used along with BkHive and Samdump2. For cracking Linux passwords john can be used with `unshadow`.

To crack Linux passwords with john use:

- `unshadow /etc/passwd /etc/shadow > hashes.txt`
- `john hashes.txt`

6 clamav

`clamav` is one of free and famous anti-viruses available for Linux distributions. Please note that although clamav is available in Linux, its primary purpsoe is not to detect Linux viruses. Even in Linux clamav is used to scan files and folders for Windows viruses. Linux is virtually virus-free. There are root-kits but `clamav` (as far as I know) does not scans for root-kits.

To scan a folder using clamscan we can use:

```
clamscan -ir <dir_name>
```

It should be noted that clamav also supports tools / daemons like freshclam, clamd etc. which periodically update clamav database from closest mirror or provide clamav daemons for protecting from viruses sent via email. Clamav daemon can be configured as milter (mail filter) in sendmail etc. SMTP servers to check incoming emails for viruses before they are sent to user Inbox.

Clamav daemon also supports options for listening on a port for receiving traffic to be scanned. If traffic received by daemon is malicious traffic then it indicates so by sending different type of response then usual on the same socket. Hence we can use clamav daemon to scan network traffic of home grown network applications too.

7 avast

I have not used `avast` on Linux platform at all. It should be very similar to `clamav` described above. Participants are encouraged to try `avast` to learn how to use it and to determine whether it has any useful features when compared to `clamav`.