Analysis of Packet Sniffers – TCPdump VS Ngrep VS Snoop

Sameer Niphadkar
IT657
Dept of Computer Science – GMU

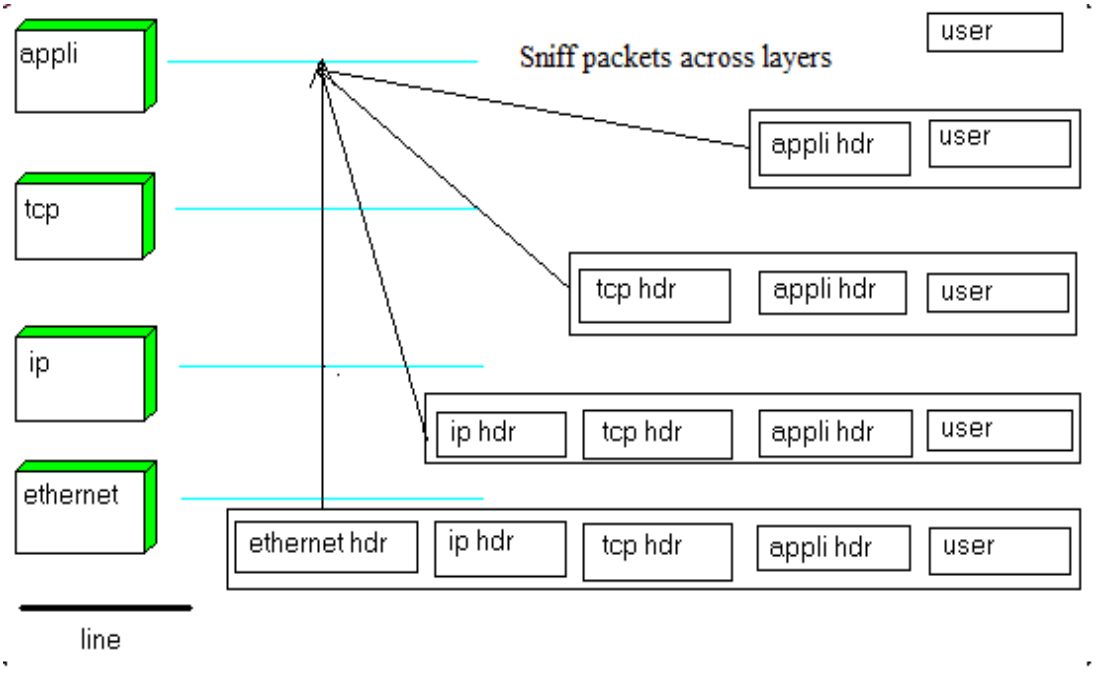
Introduction to Sniffing

In many networking protocols, transmitted data gets split into small segments, or packets, and the Internet Protocol address of the destination computer is written into the header of each packet. These packets then get passed around by routers and eventually make their way to the network segment that contains the destination computer. As each packet travels around that destination segment, the network card on each computer on the segment examines the address in the header. If the destination address on the packet is the same as the IP address of the computer, the network card grabs the packet and passes it on to its host computer.

Packet sniffers work slightly differently. Instead of just picking up the packets that are addressed to them, they set their network cards to what's known as "promiscuous mode" and grab a copy of every packet that goes past. This lets the packet sniffers see all data traffic on the network segment to which they're attached - if they're fast enough to be able to process all that mass of data, that is. This network traffic often contains very interesting information for an attacker, such as user identification numbers and passwords, confidential data - anything that isn't encrypted in some way. This data is also useful for other purposes - network engineers use packet sniffers to diagnose network faults. Hackers use packet sniffers to check for confidential data; security analysts use packet sniffers to check for hacker activity. As data streams flow across the network, the sniffer captures each packet and eventually decodes and analyzes its content according to the appropriate RFC or other specifications.

Packet Sniffers History

Packet sniffing began with the need to obtain raw packets across the several layers beneath the application protocol. Normally the packets or data frames are stripped of their headers and are passed to the upper layers. However in promiscuous mode raw packets of lower layers can directly be obtained alongside their headers. These samples are quite useful to obtain the data flow patterns across the network. This can also be used for network monitoring and management. Much of the idea behind packet sniffing began with the use of BSD packet filters (BPF) . Because network monitors run as user-level processes, packets must be copied across the kernel/user-space protection boundary. This copying can be minimized by deploying a kernel agent called a *packet filter*, which discards unwanted packets as early as possible. The BSD Packet Filter (BPF) uses a redesigned, register based filter evaluator that can be implemented efficiently on today's register based RISC CPU. BPF uses a simple, non-shared buffer model made possible by today's larger address spaces. The model is very efficient for the 'usual cases' of packet capture [BPF]. A general schematic idea of packet filter can be expressed by the figure shown below.



The architecture of BPF can be shown as below. It mainly consists of two main components : the network tap and the packet filter. The network tap collects copies of packets from the network device drivers and delivers them to listening applications. The filter decides if a packet should be accepted and, if so, how much of it to copy to the listening application.

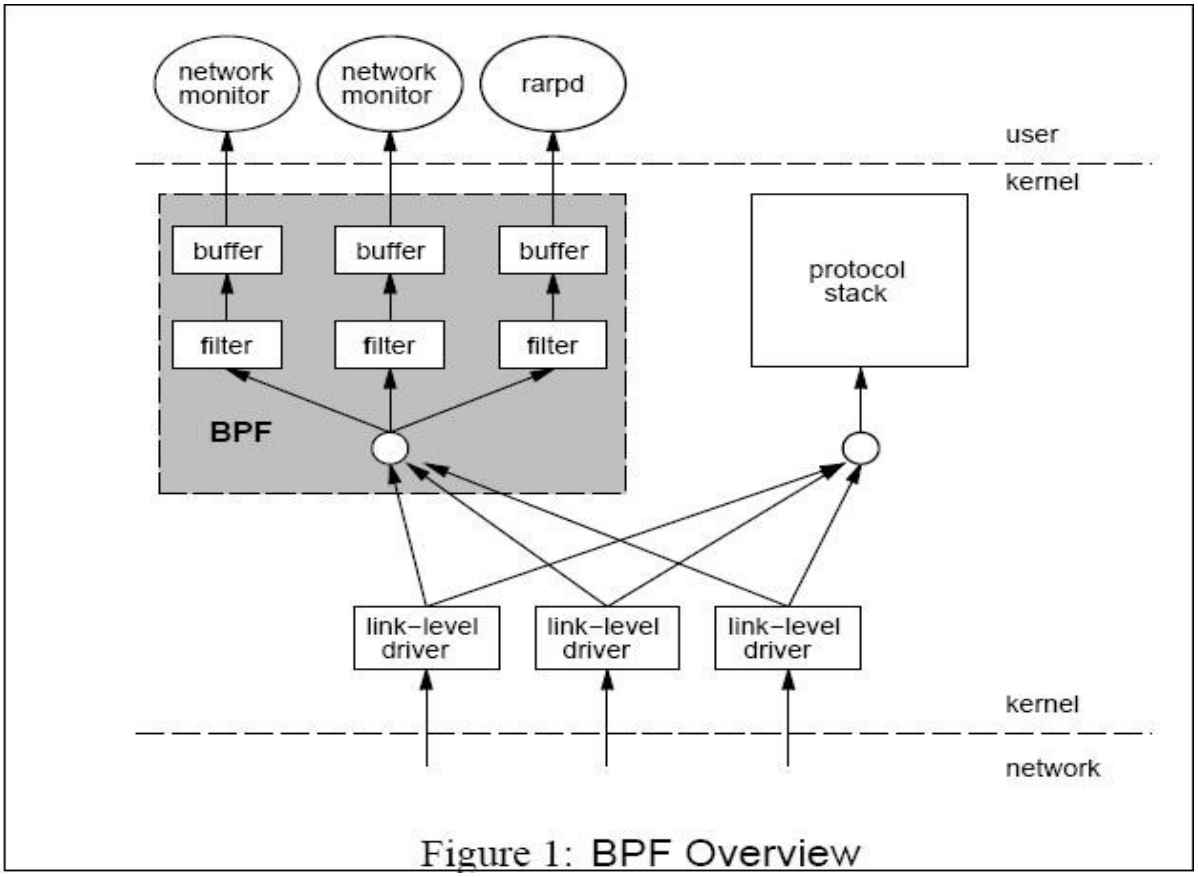


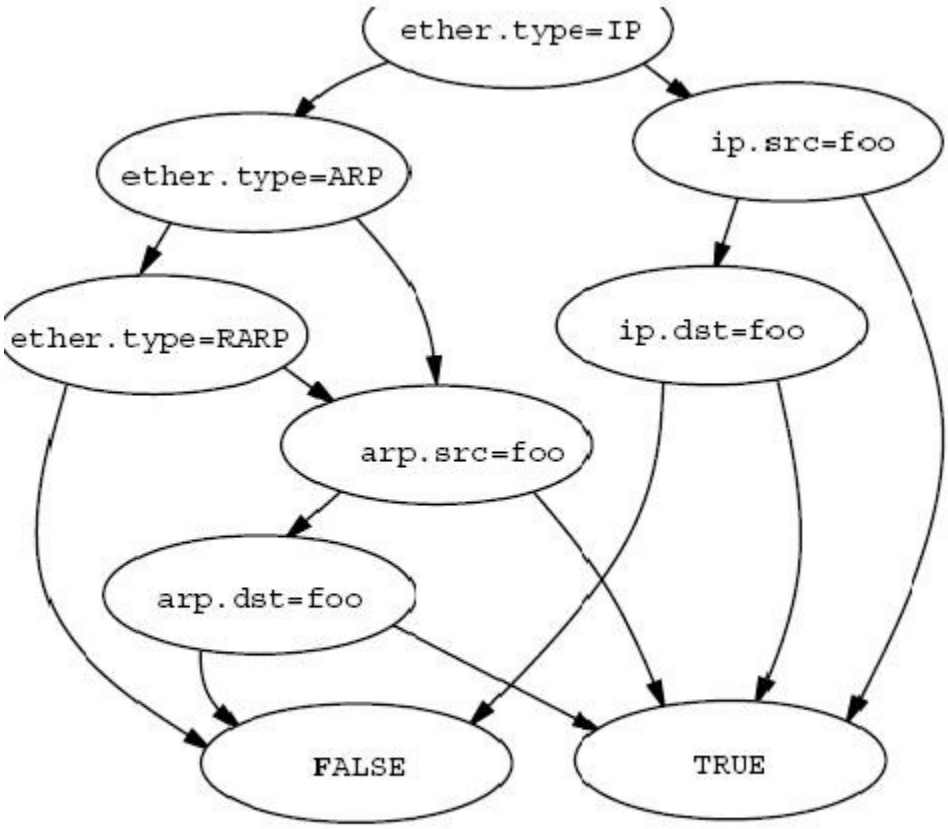
Figure 1: BPF Overview

Ref :[BPF]

When a packet arrives at a network interface the link level device driver normally sends it up the system protocol stack. But when BPF is listening on this interface, the driver first calls BPF. BPF feeds the packet to each participating process filter. This user-defined filter decides whether a packet is to be accepted and how many bytes of each packet should be saved. For each filter that accepts the packet, BPF copies the requested amount of data to the buffer associated with that filter. The device driver then regains control. If the packet was not addressed to the local host, the driver returns from the interrupt.

Since the network card is in promiscuous mode, it would accept all the packets. But most applications of a packet capture facility reject far more packets than they accept and, thus, good performance of the packet filter is critical to good overall performance. A packet filter is simply a boolean valued function on a packet. If the value of the function is *true* the kernel copies the packet for the application; if it is *false* the packet is ignored. Historically there have been two approaches to the filter abstraction: a boolean expression tree (used by CSPF) and a directed acyclic control flow graph or CFG (used by BPF).

The CFG model maps naturally into code for a register machine of all RISC based architectures. The CFG model also allows parse information to be 'built into' the flow graph. i.e., packet parse state is 'remembered' in the graph since one knows what paths one must have traversed to reach to a particular node and once a subexpression is evaluated it need not be recomputed since the control flow graph can always be (re-)organized so the value is only used at nodes that follow the original computation. Figure below shows a CFG filter function that accepts all packets with an Internet address *foo*.



CFG Filter Function for "host foo".

Ref :[BPF]

We consider a scenario where the network layer protocols are IP, ARP, and Reverse ARP, all of which contain source and destination Internet addresses. The filter should catch all cases. Accordingly, the link layer type field is tested first. In the case of IP packets, the IP host address fields are queried, while in the case of ARP packets, the ARP address fields are used. Note that once we learn that the packet is IP, we do not need to check that it might be ARP or RARP

Types of Sniffers

Since we have understood the idea and history behind network monitors or packet sniffers. It is necessary to consider the properties of an ideal packet sniffer. These can be summarized as shown below :

- View detailed IP or MAC connections statistics: IP addresses, ports, sessions, etc.
- Display TCP sessions.
- Map packets to the application that is sending or receiving them.
- View application protocols distribution, bandwidth utilization, and network nodes charts and tables.
- Ability to monitor traffic in real time as well as analyze traffic reports offline
- Ability to browse captured and decoded packets in real time.
- Ability to search for strings or hex data in captured packet contents.
- Ability to import and export packets in hex and text formats from other sniffers
- Ability to configure alarms that can notify about important events, such as suspicious packets, high bandwidth utilization, unknown addresses, etc.
- Ability to create custom plug-ins for decoding any protocol.
- Ability to exchange data with other application over TCP/IP.

Though it may not be possible to satisfy all the mentioned properties of a sniffer. Many of the modern packet sniffers do support most of them. There are different types of network sniffing tools depending on the network, application or protocol, one may be trying to monitor. There are even specialized sniffers for port scanning, promiscuous mode captures (raw packet monitoring), ARP sniffing, specialized application sniffing, encrypted mode analyzers etc. Besides, there are so many types of modern packet sniffers that it may be quite a task to consider and analyze each of them. It is because of this reason that this paper considers three of the primary and most useful packet sniffers - TCPdump, Ngrep and Snoop

TCPdump

TCPdump is the most common network debugging and packet monitoring tool that runs under the command line. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. TCPdump works on most Unix-like operating systems: Linux, Solaris, BSD, Mac OS X, HP-UX and AIX among others. In those systems, tcpdump uses the libpcap library to capture packets. The user may optionally apply a BPF-based filter to limit the number of packets seen by tcpdump; this renders the output more usable on networks with a high volume of traffic. [Wikipedia]

Tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression of the filter. It can also be run with the **-w** flag, which causes it to save the packet data to a file for later analysis, and/or with the **-r** flag, which causes it to read from a saved packet file rather than to read packets from a network interface. In all cases, only packets that match expression will be processed by tcpdump.

Tcpdump will, if not run with the -c flag, continue capturing packets until it is interrupted by a SIGINT signal (generated, for example, by typing your interrupt character, typically control-C) or a SIGTERM signal (typically generated with the kill(1) command); if run with the -c flag, it will capture packets until it is interrupted by a SIGINT or SIGTERM signal or the specified number of packets have been processed.

When tcpdump finishes capturing packets, it will report counts of:

- packets ``captured'' (this is the number of packets that tcpdump has received and processed);
- packets ``received by filter'' (the meaning of this depends on the OS on which you're running tcpdump, and possibly on the way the OS was configured - if a filter was specified on the command line, on some OS's it counts packets regardless of whether they were matched by the filter expression and, even if they were matched by the filter expression,
- packets ``dropped by kernel'' (this is the number of packets that were dropped, due to a lack of buffer space, by the packet capture mechanism in the OS)

The general format of a tcpdump protocol line is:

src > dst: flags data-seqno ack window urgent options

Src and dst are the source and destination IP addresses and ports. Flags are some combination of S (SYN), F (FIN), P (PUSH), R (RST), W (ECN CWR) or E (ECN-Echo), or a single `.' (no flags). Data-seqno describes the portion of sequence space covered by the data in this packet (see example below). Ack is sequence number of the next data expected the other direction on this connection. Window is the number of bytes of receive buffer space available the other direction on this connection. Urg indicates there is `urgent' data in the packet. Options are tcp options enclosed in angle brackets (e.g., <mss 1024>).

Src, dst and flags are always present. The other fields depend on the contents of the packet's tcp protocol header and are output only if appropriate.

Here is the opening portion of an rlogin from host rtsg to host csam.

```
  rtsg.1023 > csam.login: S 768512:768512(0) win 4096 <mss 1024>
    csam.login > rtsg.1023: S 947648:947648(0) ack 768513 win 4096 <mss
1024>
    rtsg.1023 > csam.login: . ack 1 win 4096
    rtsg.1023 > csam.login: P 1:2(1) ack 1 win 4096
    csam.login > rtsg.1023: . ack 2 win 4096
    rtsg.1023 > csam.login: P 2:21(19) ack 1 win 4096
    csam.login > rtsg.1023: P 1:2(1) ack 21 win 4077
    csam.login > rtsg.1023: P 2:3(1) ack 21 win 4077 urg 1
    csam.login > rtsg.1023: P 3:4(1) ack 21 win 4077 urg 1
```

The first line says that tcp port 1023 on rtsg sent a packet to port login on csam. The S indicates that the SYN flag was set. The packet sequence number was 768512 and it contained no data. (The notation is `first:last(nbytes)' which means `sequence numbers first up to but not including last which is nbytes bytes of user data'.) There was no piggy-backed ack, the available receive window was 4096 bytes and there was a max-segment-size option requesting an mss of 1024 bytes.

Csam replies with a similar packet except it includes a piggy-backed ack for rtsg's SYN. Rtsg then acks csam's SYN. The `.' means no flags were set. The packet contained no data so there is no data sequence number. Note that the ack sequence number is a small integer (1). The first time tcpdump sees a tcp `conversation', it prints the sequence number from the packet. On subsequent packets of the conversation, the difference between the current packet's sequence number and this initial sequence number is printed. This means that sequence numbers after the first can be interpreted as relative byte positions in the conversation's data stream (with the first data byte each direction being `1'). `-S' will override this feature, causing the original sequence numbers to be output.

On the 6th line, rtsg sends csam 19 bytes of data (bytes 2 through 20 in the rtsg -> csam side of the conversation). The PUSH flag is set in the packet. On the 7th line, csam says it's received data sent by rtsg up to but not including byte 21. Most of this data is apparently sitting in the socket buffer since csam's receive window has gotten 19 bytes smaller. Csam also sends one byte of data to rtsg in this packet. On the 8th and 9th lines, csam sends two bytes of urgent, pushed data to rtsg.

[Tcpdump man page]

Tcpdump is frequently used to debug applications that generate or receive network traffic. It can also be used for debugging the network setup itself, by determining whether all necessary routing is occurring properly, allowing the user to further isolate the source of a problem. It is also possible to use tcpdump for the specific purpose of intercepting and displaying the communications of another user or computer. A user with the necessary privileges on a system acting as a router or gateway through which unencrypted traffic such as TELNET or HTTP passes can use tcpdump to view login IDs, passwords, the URLs and content of websites being viewed, or any other unencrypted information.

It can thus be observed that TCPdump is a fairly simple and easy to use utility which can used for any general packet monitoring mechanism in promiscuous mode. However there are few disadvantages with TCPdump –

1. Only raw dump file created – i.e no measures adopted to observe specific activities by large
2. Limitation on the type of network traffic that can be analyzed i.e only TCP based protocols can be seen,
3. It is a fairly manual implementation with the user required to know all the options for screening specific packets. - No easy user interface
4. Packets blocked by a gateway firewall may not be seen
5. External implementations like user defined network condition analysis requires use of other tools like tcppreplay or tcpopera alongside

Ngrep

Ngrep strives to provide most of GNU grep's common features, applying them to the network layer. Ngrep is a pcap-aware tool that will allow you to specify extended regular or hexadecimal expressions to match against data payloads of packets. It currently recognizes IPv4/6, TCP, UDP, ICMPv4/6, IGMP and Raw across Ethernet, PPP, SLIP, FDDI, Token Ring and null interfaces, and understands

BPF filter logic in the same fashion as more common packet sniffing tools, such as tcpdump and snoop. A "network grep" system is based around raw packet capture pumped through a "regular expression" parser that finds patterns in the network traffic. An example pattern would be: "/cgi-bin/phf", which would indicate an attempt to exploit the vulnerable CGI script called "phf". Once building such a system, one can then analyze well-known attacks, extract strings specific to those attacks, and add them to database of patterns.

"Regexp" (regular expression) is a common pattern-matching language in the UNIX environment. While it has traditionally been used for searching text files, it can also be used for arbitrary binary data. In truth, such systems have more flexible matching criteria, such as finding ports or matching TCP flags. "libpcap" (library for packet capture) is a common library available for UNIX systems that "sniffs" packets off a wire. Most UNIX-based intrusion detection systems (of any kind) use libpcap, though many also have optimized drivers for a small subset of platforms.

Ngrep may simply feed the output of libpcap (or tcpdump) into the regular expression parse, where the expressions come from a file on the disk. Some even simpler systems don't even use regular expressions and simply compare packets with well-known byte patterns. Ngrep has traditionally been used to debug plain text protocol interactions such as HTTP, SMTP, FTP, etc., to identify and analyze anomalous network communications such as those between worms, viruses and/or zombies, and to store, read and reprocess pcap dump files while looking for specific data patterns. On the other hand, it can be used to do the more mundane plain text credential collection as with HTTP Basic Authentication, FTP or POP3 authentication, and so forth. Like all useful tools, it can be used for good and bad.

Basic packet sniffing is easy with ngrep. It supports BPF filter logic, which means to say constraining what ngrep sees and displays is very easy. For e.g

```
ngrep -d any port 25
```

Monitor all activity crossing source or destination port 25 (SMTP).

```
ngrep -d any 'error' port syslog
```

Monitor any network-based syslog traffic for the occurrence of the word ``error''. ngrep knows how to convert service port names (on UNIX, located in ``/etc/services'') to port numbers.

```
 ngrep -W byline port 80
interface: eth0 (64.90.164.72/255.255.255.252)
filter: ip and ( port 80 )
####
T 67.169.59.38:42177 -> 64.90.164.74:80 [AP]
GET / HTTP/1.1.
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; X11; Linux i686) Opera ...
Host: www.darkridge.com.
Accept: text/html, application/xml;q=0.9, application/xhtml+xml;q=0.9 ...
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1.
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0.
Cookie: SQMSESSID=5272f9ae21c07eca4dfd75f9a3cda22e.
Cookie2: $Version=1.
Cache-Control: no-cache.
Connection: Keep-Alive, TE.
TE: deflate, gzip, chunked, identity, trailers.
.
```

```
##
T 64.90.164.74:80 -> 67.169.59.38:42177 [AP]
HTTP/1.1 200 OK.
Date: Mon, 29 Mar 2004 00:47:25 GMT.
Server: Apache/2.0.49 (Unix).
Last-Modified: Tue, 04 Nov 2003 12:09:41 GMT.
ETag: "210e23-326-f8200b40".
Accept-Ranges: bytes.
Vary: Accept-Encoding,User-Agent.
Content-Encoding: gzip.
Content-Length: 476.
Keep-Alive: timeout=15, max=100.
Connection: Keep-Alive.
Content-Type: text/html; charset=ISO-8859-1.
Content-Language: en.
.
.........}S]..0.|..........H...8........@..\....(.....Dw.%.,..;.k.. ...
.;kw*U.j.<...\0Tn.l.:......>Fs....'....h.'...u.H4..'.6.vIDI.......N.r ...
..H..#..J....u.?..]....^..2.....e8v/gP.....].48...qD!..........#y...m ...
####
```

(Content visually truncated for display purposes.)

``-W byline" mode tells ngrep to respect embedded line feeds when they occur. You'll note from the output above that there is still a trailing dot (``.") on each line, which is the carriage-return portion of the CRLF pair. Using this mode, now the output has become much easier to visually parse.

```
# ngrep -w 'm'  -I /tmp/dns.dump
input: /tmp/dns.dump
match: ((^m\W)|(\Wm$)|(\Wm\W))
#
U 203.115.225.24:53 -> 64.90.164.74:53
  ............m.razor2.cloudmark.com.......)........
#
U 64.90.164.74:53 -> 203.115.225.24:53
  ............m.razor2.cloudmark.com..............'.ns1...hostmaster..ws
..
  ..p.... ..:.......)........
##exit
```

Above we searched for the letter ``m", matched as a word (``-w"). This yields two packets.

```
# ngrep -tD ns3  -I /tmp/dns.dump
input: /tmp/dns.dump
match: ns3
####
U 2004/03/28 20:32:37.088525 64.90.164.74:53 -> 195.113.155.7:2949
  ............a.razor2.cloudmark.com.................agony...4.........B
..
  ............ns1.............ns2.............ns3...X.........@Z.J.j
..
  ........@Z...|.........B..;
exit
```

Here we've added ``-t'' which means print the absolute timestamp on the packet, and ``-D'' which means replay the packets by the time interval at which they were recorded. The latter is a feature for observing the traffic at the rates/times they originally seen, though in this example it's not terribly effective as there is only one packet being matched. There are other types of advanced Ngrep features that provide matching with various types of regular expressions. For e.g - the following regular expression when prompted would get back all of the hits with the strings c+dir+c and Directory of c.

```
# ngrep -w 'c'  -I /tmp/dns.dump
input: /tmp/dns.dump
match: (([C|c][+][D|d][I|i][R|r][+][C|c]))
```

To view all packets that contain the string GET (presumably HTTP requests),

```
# ngrep -q GET
```

Pattern matches can be constrained further to match particular protocols, ports, or other criteria using BPF filters. This is the filter language used by common packet sniffing tools, such as tcpdump and snoop. To view GET or POST strings sent to destination port 80, use this command line:

```
# ngrep -q 'GET|POST' port 80
```

By using ngrep creatively, one can detect anything from virus activity to spam email. We can thus easily see that ngrep has an advantage over other protocol sniffing tools. Because ngrep does not have pre-conceived notion about what network traffic is supposed to look like, it can often detect attacks that other tools might miss. For example, if a company is running a POP3 server on a different port, it is likely that protocol sniffing tool will not recognize the traffic as POP3. Therefore, any attacks against the port might go undetected. On the other hand, a network-grep style system doesn't necessarily care about port numbers and will check for the same signatures regardless of ports. However ngrep also has some disadvantages like :

1. Ngrep based systems result in larger numbers of false positives. For e.g - alarms to go off if there is a match with the regular expression even when no real threat is present.
2. Pure packet sniffers are often able to run faster because a protocol decode doesn't have to "search" a frame.

Snoop

Snoop is a very flexible command line packet sniffer included as part of Sun Microsystems' Solaris Operating System. It is pretty cable sniffer equal or better then TCPdump. Snoop file format was is different from PCAP and was defined in RFC 1761. Snoop displays packets in a single-line summary form or in verbose multi-line forms. It can display packets as soon as they are received or saved to a file. When snoop writes to an intermediate file, packet loss under busy trace conditions is unlikely. Snoop itself can be used read and interpret the file e.g - print summary expended summary and full packets dumps. It has pretty powerful packet filtering engine

The snoop command can capture both IPv4 and IPv6 packets. It can display IPv6 headers, IPv6 extension headers, ICMPv6 headers, and neighbor discovery protocol data. By default, the snoop command displays both IPv4 and IPv6 packets. IPv6 traffic snoop capabilities are very similar to tcpdump and output formats are almost identical. Still there are some differences and for example the IDS Snort can read tcpdump binary files, but not snoop binary files. Another tool Ethereal's editcap program can be used to convert the snoop file to a tcpdump file.

Only packets for which the expression is true will be selected. If no expression is provided it is assumed to be true. Given a filter expression, snoop generates code for either the kernel packet filter or for its own internal filter. If capturing packets with the network interface, code for the kernel packet filter is generated. This filter is implemented as a streams module, upstream of the buffer module. The buffer module accumulates packets until it becomes full and passes the packets on to snoop. The kernel packet filter is very efficient, since it rejects unwanted packets in the kernel before they reach the packet buffer or snoop. The kernel packet filter has some limitations in its implementation; it is possible to construct filter expressions that it cannot handle. In this event, snoop tries to split the filter and do as much filtering in the kernel as possible. The remaining filtering is done by the packet filter for snoop. The -C flag can be used to view generated code for either the packet filter for the kernel or the packet filter for snoop. If packets are read from a capture file using the -i option, only the packet filter for snoop is used.

A filter expression consists of a series of one or more boolean primitives that may be combined with boolean operators (AND, OR, and NOT). Normal precedence rules for boolean operators apply. Order of evaluation of these operators may be controlled with parentheses. Since parentheses and other filter expression characters are known to the shell, it is often necessary to enclose the filter expression in quotes. There are two forms in which snoop can display

Summary form – Only the application level protocol is displayed. For e.g - for an NFS packet only NFS information displayed. All underlying RPC, UDP, IP, and Ethernet frame information is suppressed

Verbose form – Display everything

```
example# snoop -i pkts -p 99,108 #Capture packets in promiscuous mode from
99 to 108
 99   0.0027   boutique -> sunroof    NFS C GETATTR FH=8E6
100   0.0046   sunroof -> boutique    NFS R GETATTR OK
101   0.0080   boutique -> sunroof    NFS C RENAME FH=8E6C MTra00192 to
.nfs08
102   0.0102   marmot -> viper        NFS C LOOKUP FH=561E screen.r.13.i386
103   0.0072   viper -> marmot        NFS R LOOKUP No such file or
directory
104   0.0085   bugbomb -> sunroof     RLOGIN C PORT=1023 h
105   0.0005   kandinsky -> sparky    RSTAT C Get Statistics
106   0.0004   beeblebrox -> sunroof  NFS C GETATTR FH=0307
107   0.0021   sparky -> kandinsky    RSTAT R
108   0.0073   office -> jeremiah     NFS C READ FH=2584 at 40960 for 8192

To look at packet 101 in more detail

example# snoop -i pkts -v -p101
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 101 arrived at 16:09:53.59
ETHER:  Packet size = 210 bytes
ETHER:  Destination = 8:0:20:1:3d:94, Sun
ETHER:  Source      = 8:0:69:1:5f:e,  Silicon Graphics
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4, header length = 20 bytes
IP:   Type of service = 00
```

```
IP:          ..0. .... = routine
IP:          ...0 .... = normal delay
IP:          .... 0... = normal throughput
IP:          .... .0.. = normal reliability
IP:    Total length = 196 bytes
IP:    Identification 19846
IP:    Flags = 0X
IP:    Fragment offset = 0 bytes
IP:    Time to live = 255 seconds/hops
IP:    Protocol = 17 (UDP)
IP:    Header checksum = 18DC
IP:    Source address = 172.16.40.222, boutique
IP:    Destination address = 172.16.40.200, sunroof
UDP:   ----- UDP Header -----
UDP:
UDP:   Source port = 1023
00800002046314AFC450000
NFS:   File name = .nfs08
NFS:
UDP:   Destination port = 2049 (Sun RPC)
UDP:   Length = 176
UDP:   Checksum = 0
UDP:
RPC:   ----- SUN RPC Header -----
RPC:
RPC:   Transaction id = 665905
RPC:   Type = 0 (Call)
RPC:   RPC version = 2
RPC:   Program = 100003 (NFS), version = 2, procedure = 1
RPC:   Credentials: Flavor = 1 (Unix), len = 32 bytes
RPC:      Time = 06-Mar-90 07:26:58
RPC:      Hostname = boutique
RPC:      Uid = 0, Gid = 1
RPC:      Groups = 1
RPC:   Verifier   : Flavor = 0 (None), len = 0 bytes
RPC:
NFS:   ----- SUN NFS -----
NFS:
NFS:   Proc = 11 (Rename)
NFS:   File handle = 000016430000000100080000305A1C47
```

As it can be seen snoop is a more efficient and useful monitoring tool compared to raw TCPdump since is has a much better user interface and displaying capabilities, besides having all the advantages of TCPdump. The only disadvantage being that since it is tightly integrated within the Solaris kernel, its use is largely limited to Sun based systems. Though there have been hacks to port it to other kernels like FreeBSD and Linux, most of them have been fairly limited by number.

Conclusion

We can conclude by summing that each of the above mentioned tool has its uniqueness and the best way to enhance security is to combine their usage with other scanners, IDS and network management systems.

References

Steven McCanne and Van Jacobso " The BSD Packet Filter: A New Architecture for User-level Packet Capture" Lawrence Berkeley Laboratory 1992

Ashish Chaurasia "Network packet capturing for Linux ", IBM Corp

Naveed Afzal "Host Fingerprinting and Firewalking With hping"

Fiona Wong and S. Felix Wu "TCP-Opera" University of California, Davis

Ryan Spangler "Analysis of Remote Active Operating System Fingerprinting Tools" University of Michigan

http://www.packet-sniffer.net/network-security.htm

http://www.eff.org/wp/detecting-packet-injection

Wikipedia – TCPdump, Ngrep, Snoop

http://www.geocities.com/amit_saha_works/linux/html/sniff.html

http://search.linuxsecurity.com/resource_files/intrusion_detection/network-intrusion-detection.html#4.3

http://www.softpanorama.org/Net/Network_security/Sniffers/snoop.shtml

http://docs.sun.com/app/docs/doc/819-2240/snoop-1m?&a=view&q=snoop

http://www.ethereal.com/docs/man-pages/tcpdump.8.html