TNE30019/TNE80014 – Unix for Telecommunications

Network and Traffic Analysis Tools – tcpdump

Dr. Jason But

Swinburne University

Outline

- Network Sniffing
  - Why
  - How is it implemented in kernel
- PCAP library
- tcpdump
  - Usage
  - Post Processing
- Alternative PCAP-based utilities

Network Sniffing

- Why sniff packets off a network
  - Examine network activity
  - Check what traffic is being generated
  - Debug network problems
  - Determine correctness of protocol implementations
  - Generate network statistics

Useful for

- Real network deployment/management
- Research purposes

Network Sniffing

- Biggest difficulties include
  - Sniffing traffic is platform dependent
  - Has to be handled by kernel

Regular network traffic reception (kernel)

1. Packets arrive at Network Interface Card (NIC)
2. OS is interrupted
3. Device driver reads packet(s)
4. Strip link-layer headers
5. Pass packets to OS
6. Check network layer headers (IP) and passes to IP stack
7. Check if packet should go to other router or local process
8. IP stack checks protocol and passes to TCP/UDP stack
9. Eventually pass to application via sockets API

Network Sniffing

Extra tasks when capturing traffic

- Packet are delivered to sniffing application regardless of packet details
- Packet arrival timestamp is noted
- Other meta information is noted (e.g. packet length)
- Packet needs to be (partially) copied to get unique instance
  - Differs from default, where pointer to packet is passed around
  - Packet has to be copied again from kernel memory to user memory – copy and meta info (e.g. timestamp) passed to capture application

- This is **NOT** standardised!!

Packet Capture (PCAP) Library

- API to capture packets is different on different platforms
  - Linux – Packet Filter Sockets
  - BSD – BPF (Berkeley Packet Filter)
- **PCAP**[1] provides common API on top of different systems

PCAP Features (C Library)

- Opening and reading from capture device
- Specifying filters – only receive packets that pass filter
- Writing/reading of captured data from/to PCAP-format file
- Originally written as part of **tcpdump**

[1] `http://www.tcpdump.org`

`tcpdump`

- Capture application using **PCAP** library
- Also available at `http://www.tcpdump.org`

Installation – FreeBSD

Port: `/usr/ports/net/tcpdump`
OR
`pkg install tcpdump`

- Will automatically download and install **PCAP** (port: `/usr/ports/net/libpcap`)
- Command line application to capture packets *"off the wire"*

`tcpdump` – Options

Default Options

- Captures first 68 bytes of each packet (FreeBSD)
  - Ethernet Frame – 14 bytes
  - IP Header – 20 bytes
  - TCP Header – 20 bytes plus size of TCP options
  - Allows to analyse IP and transport protocol headers
- Prints information about captured packets as text to stdout

Other Options

- Capture all bytes
- Write to file for later post-processing
- Read from file rather than live capture
- Post-filter packets
- Verbose output (many levels)
- Different timestamp outputs

Post Processing Captured Traffic

- Why would we do this?
- Don't need real-time processing
- Can't do real-time processing (complex processing)
- Processing may require data collected over long time window or from different locations
- Many **PCAP**-enabled programs to process packets – often specialise on certain analysis (e.g. TCP analysis)

tcpdump – Example Output

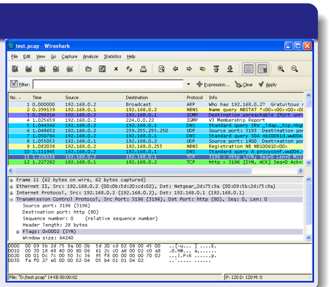
Line breaks added to wrap-around long lines

```
> tcpdump -w test.dmp
> tcpdump -nSr test.dmp port 80
reading from file test.dmp, link-type EN10MB (Ethernet)
12:36:44.995040 IP 136.186.229.100.39219 > 8.8.178.110.80:
   Flags [S], seq 559979967, win 14400, options
   [mss 1440,sackOK,TS val 2359415211 ecr 0,nop,wscale 7], length 0
12:36:45.153082 IP 8.8.178.110.80 > 136.186.229.100.39219:
   Flags [S.], seq 621286651, ack 559979968, win 14080, options
   [mss 1380,sackOK,TS val 467405075 ecr 2359415211,nop,wscale 8], length 0
12:36:45.153118 IP 136.186.229.100.39219 > 8.8.178.110.80:
   Flags [.], ack 1, win 113, options
   [nop,nop,TS val 2359415369 ecr 467405075], length 0
12:36:45.153245 IP 136.186.229.100.39219 > 8.8.178.110.80:
   Flags [P.], seq 559979968:559980430, ack 621286652, win 113, options
   [nop,nop,TS val 2359415369 ecr 467405075], length 462
```

Other PCAP-based Applications

Wireshark

- GUI-based packet capture
- Provides some analysis tools
- `http://www.wireshark.org`



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.2	Broadcast	ARP	Who has 192.168.0.2? Gratuitous A
2	0.299139	192.168.0.1	192.168.0.2	NBNS	Name query NBSTAT *<00><00><00><0
3	0.299214	192.168.0.2	192.168.0.1	ICMP	Destination unreachable (Port unr
4	1.025659	192.168.0.2	224.0.0.22	IGMP	V3 Membership Report
5	1.044366	192.168.0.2	192.168.0.1	DNS	Standard query SRV _ldap._tcp.nbg
6	1.048652	192.168.0.2	239.255.255.250	UDP	Source port: 3193 Destination po
7	1.050784	192.168.0.2	192.168.0.1	DNS	Standard query SOA nb10061d.ww004.
8	1.055053	192.168.0.1	192.168.0.2	UDP	Source port: 1900 Destination po
9	1.082038	192.168.0.2	192.168.0.255	NBNS	Registration NB NB10061D<00>
10	1.111945	192.168.0.2	192.168.0.1	DNS	Standard query A proxyconf.ww004.
11	1.226156	192.168.0.2	192.168.0.1	TCP	3196 > http [SYN] Seq=0 Len=0 MSS
12	1.227282	192.168.0.1	192.168.0.2	TCP	http > 3196 [SYN, ACK] Seq=0 Ack=

```
⊞ Frame 11 (62 bytes on wire, 62 bytes captured)
⊞ Ethernet II, Src: 192.168.0.2 (00:0b:5d:20:cd:02), Dst: Netgear_2d:75:9a (00:09:5b:2d:75:9a)
⊞ Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
⊟ Transmission Control Protocol, Src Port: 3196 (3196), Dst Port: http (80), Seq: 0, Len: 0
    Source port: 3196 (3196)
    Destination port: http (80)
    Sequence number: 0    (relative sequence number)
    Header length: 28 bytes
  ⊞ Flags: 0x0002 (SYN)
    Window size: 64240
```

```
0000  00 09 5b 2d 75 9a 00 0b  5d 20 cd 02 08 00 45 00   ..[-u... ] ....E.
0010  00 30 18 48 40 00 80 06  61 2c c0 a8 00 02 c0 a8   .0.H@... a,......
0020  00 01 0c 7c 00 50 3c 36  95 f8 00 00 00 00 70 02   ...|.P<6 ......p.
0030  fa f0 27 e0 00 00 02 04  05 b4 01 01 04 02         ..'..... ......
```

File: "D:/test.pcap" 14 KB 00:00:02 P: 120 D: 120 M: 0

NetSniff

- Developed at Swinburne
- Reconstructs TCP flows
- Generates application-layer statistics
- Extended sniffing with rolling logs
- `http://caia.swin.edu.au/ice/tools/netsniff`