Como utilizar o tcpdump

Ricardo Iramar dos Santos - Agent Smith
13/10/2008 - Versão 0.1

1. Introdução

Eu estava pensando em escrever aqui uma definição para o *tcpdump* mas tenho a certeza que está descrita aqui http://en.wikipedia.org/wiki/Tcpdump é o suficiente para a introdução desta documentação.

Esta documentação não irá descrever como instalar o *tcpdump*, até mesmo porque a maioria das distribuições linux possuem binários ou é extremamente fácil a sua instalação. Em todo caso você pode visitar http://www.tcpdump.org que com certeza vai encontrar algo a respeito.

No final desta documentação você estará apto a usar o *tcpdump* com filtros como: endereço *ip* ou porta de origem ou destino, endereço de rede, tipo de protocolo e até mesmo filtrar especificando uma *flag* do cabeçalho TCP.

2. Utilização

O *tcpdump* necessariamente não precisa de um parâmetro para ser executado, você só precisa estar como *root* para poder executá-lo pois o *tcpdump* irá precisar colocar a sua interface em *promiscuous mode*. Calma! Isso não tem nada ver com a promiscuidade que você esta pensando, para saber do que se trata visite http://en.wikipedia.org/wiki/Promiscuous_mode.

Matando dois coelhos com uma única tijolada vamos ver um exemplo do *tcpdump* sem parâmetros e aproveitamos para entender sua saída padrão também:

```
smith ~ # tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
...
21:02:02.579320 IP 192.168.0.1.1921 > smith.zion.ssh: S 4011627552:4011627552(0) win 64512 <mss 1460,nop,nop,sackOK>
21:02:02.579462 IP smith.zion.ssh > 192.168.0.1.1921: S 2154231898:2154231898(0) ack 4011627553 win 5840 <mss 1460,nop,nop,sackOK>
21:02:02.579707 IP 192.168.0.1.1921 > smith.zion.ssh: . ack 1 win 64512
21:02:02.623121 IP smith.zion.ssh > 192.168.0.1.1921: P 1:21(20) ack 1 win 5840
21:02:02.623576 IP 192.168.0.1.1921 > smith.zion.ssh: P 1:29(28) ack 21 win 64492
...
327 packets captured
1038 packets received by filter
266 packets dropped by kernel
```

Como não informamos qual interface queríamos monitorar ele pegou a primeira disponível, no meu caso a eth0. Podemos notar isso no trecho ***listening on eth0***. Para selecionar a interface na qual se deseja capturar os pacotes basta usar o parâmetro "**-i nome_da_interface**", como por exemplo "**-i eth1**".

Na saída padrão os campos estão separados por um espaço em branco. Como pode ser facilmente notado, o primeiro campo é o que eles chamam de *timestamp*, para nós é o horário em que o pacote foi capturado.

O segundo campo informa que o tipo do pacote ethernet foi capturado, no nosso caso é o *ip*. Poderia ser também *ip6*, *arp*, *rarp*, *atalk*, *aarp*, *decnet*, *sca*, *lat*, *mopdl*, *moprc*, *iso*, *stp*, *ipx* ou *netbeui*, mas isso não tem a mínima importância agora.

O terceiro campo são dois em um, famoso "endereço_de_origem.porta_de_origem". No nosso caso, para o primeiro pacote descrito no exemplo, o endereço de origem é "192.168.0.1" e a porta de origem "1921".

Bem, o quarto campo não é bem um campo mas somente um sinal para indicar o sentido do pacote. Desta forma não tem como a gente confundir origem com destino.

Idêntico ao terceiro o quinto campo é "endereço_de_destino.porta_de_destino". Como não especificamos nenhum parâmetro o *tcpdump* converteu o endereço de destino para o nome *smith.zion* e a porta "22" para *ssh*. Se você não quiser que o *tcpdump* fique convertendo os endereços e portas basta utilizar o parâmetro "-n". Esse campo sempre termina com ":", não sei a sua utilização mas deve ter uma razão lógica para isso, acho que só perguntando para os desenvolvedores do *tcpdump*.

O quinto campo é referente ao bit de controle, no caso do primeiro pacote "S" quer dizer que é um pacote do tipo SYN (*Synchronize*). Esse campo poderia ser "R" (*Reset*), "F" (*Finish*), "P" (*Push*), etc. Para saber mais detalhadamente leia a RFC 793 (*TRANSMISSION CONTROL PROTOCOL*).

Os demais campos não nos interessa neste momento.

Agora que você já sabe o básico da saída padrão vamos ver um exemplo bem básico. Digamos que você queira pegar somente os pacotes que estão sendo enviados ou recebidos para www.gooogle.com.br. Para isso basta utilizar a expressão "**host**" seguido do "**nome do host**" que deseja filtrar.

```
smith ~ # tcpdump -i eth0 host www.google.com.br
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:18:11.483790 IP 192.168.0.2.49286 > br-in-f104.google.com.http: S 3414458403:3414458403(0) win 5840 <mss 1460,sackOK,timestamp 226675 0,nop,w
10:18:11.495372 IP br-in-f104.google.com.http > 192.168.0.2.49286: S 2198482267:2198482267(0) ack 3414458404 win 5672 <mss 1430,sackOK,timestamp
10:18:11.495555 IP 192.168.0.2.49286 > br-in-f104.google.com.http: . ack 1 win 46 <nop,nop,timestamp 226678 49295402>
```

Mas cade www.google.com.br? Calma, o google tem vários servidores e *br-in-f104.google.com* é somente mais um deles. Mas acredite no tcpdump, ele não está mentindo e como geralmente queremos saber qual o ip e o número da porta vamos executar agora com o parâmetro "**-n**" para ver o que acontece.

```
smith ~ # tcpdump -i eth0 -n host www.google.com.br
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
13:17:00.795873 IP 192.168.0.2.58605 > 209.85.193.104.80: S 4145938788:4145938788(0) win 5840 <mss 1460,sackOK,timestamp 2909130 0,nop,wscale 7>
13:17:00.807891 IP 209.85.193.104.80 > 192.168.0.2.58605: S 4200328202:4200328202(0) ack 4145938789 win 5672 <mss 1430,sackOK,timestamp 46584142
13:17:00.808066 IP 192.168.0.2.58605 > 209.85.193.104.80: . ack 1 win 46 <nop,nop,timestamp 2909133 46584142>
```

Agora não temos mais nomes somente números facilitando a leitura e entendimento. Você também pode usar ips na expressão ao invés do nome do host como por exemplo:

```
smith ~ # tcpdump -i eth0 -n host 192.168.0.1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
13:19:50.636347 IP 192.168.0.2 > 192.168.0.1: ICMP echo request, id 59421, seq 1, length 64
13:19:50.636652 IP 192.168.0.1 > 192.168.0.2: ICMP echo reply, id 59421, seq 1, length 64
13:20:43.622909 IP 192.168.0.2.54940 > 192.168.0.1.23: S 3347765906:3347765906(0) win 5840 <mss 1460,sackOK,timestamp 2964839 0,nop,wscale 7>
```

```
13:20:43.623236 IP 192.168.0.1.23 > 192.168.0.2.54940: R 0:0(0) ack 3347765907 win 0
13:20:47.828000 IP 192.168.0.2.54941 > 192.168.0.1.23: S 3403459962:3403459962(0) win 5840 <mss 1460,sackOK,timestamp 2965890 0,nop,wscale 7>
13:20:47.828321 IP 192.168.0.1.23 > 192.168.0.2.54941: R 0:0(0) ack 3403459963 win 0
13:20:48.620885 arp who-has 192.168.0.1 tell 192.168.0.2
13:20:48.621120 arp reply 192.168.0.1 is-at 00:19:5b:b6:ff:59
13:21:01.091548 IP 192.168.0.2.44304 > 192.168.0.1.80: S 3619763307:3619763307(0) win 5840 <mss 1460,sackOK,timestamp 2969206 0,nop,wscale 7>
```

Mas eu queria somente o acesso via HTTP (porta 80). Sem problemas basta usar a expressão "**host**" em conjunto com "**port**" utilizando o operador "**and**", veja como:

```
smith ~ # tcpdump -i eth0 -n host 192.168.0.1 and port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
13:32:03.548794 IP 192.168.0.2.36993 > 192.168.0.1.80: S 1122631006:1122631006(0) win 5840 <mss 1460,sackOK,timestamp 3134828 0,nop,wscale 7>
13:32:03.549074 IP 192.168.0.1.80 > 192.168.0.2.36993: S 12860500:12860500(0) ack 1122631007 win 5840 <mss 1460>
13:32:03.549136 IP 192.168.0.2.36993 > 192.168.0.1.80: . ack 1 win 5840
```

Você pode concatenar quantos filtros forem precisos basta usar os operadores "**and**" ou "**or**" a cada novo filtro. Esses operadores também podem serem utilizados em conjunto com o operador "**not**" no caso de uma lógica inversa.Na man page do tcpdump (http://www.tcpdump.org/tcpdump_man.html) você irá encontrar todos os filtros possíveis, vale a pena dar uma olhada. Vejamos um outro exemplo:

```
smith ~ # tcpdump -i eth0 -n dst host 192.168.0.1 and not port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
10:50:37.376405 arp who-has 192.168.0.1 tell 192.168.0.2
10:51:01.670855 IP 192.168.0.2 > 192.168.0.1: ICMP echo request, id 57367, seq 1, length 64
10:51:02.671324 IP 192.168.0.2 > 192.168.0.1: ICMP echo request, id 57367, seq 2, length 64
10:53:55.283902 IP 192.168.0.2.59910 > 192.168.0.1.23: S 3501945306:3501945306(0) win 5840 <mss 1460,sackOK,timestamp 192212 0,nop,wscale 7>
```

Neste exemplo capturamos somente os pacotes com destino ao host 192.168.0.1 exceto com destino a porta 80. Perceba que por utilizarmos somente o filtro "**dst host**" foram capturados os pacotes somente em um sentido indicado por ">". Vejamos outro exemplo básico especificando o tipo de protocolo e a rede:

```
smith ~ # tcpdump -i eth0 -n net 192.168 and icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:21:48.518096 IP 192.168.0.2 > 192.168.0.1: ICMP echo request, id 60206, seq 1, length 64
11:21:48.518404 IP 192.168.0.1 > 192.168.0.2: ICMP echo reply, id 60206, seq 1, length 64
11:21:49.517104 IP 192.168.0.2 > 192.168.0.1: ICMP echo request, id 60206, seq 2, length 64
```

Neste exemplo capturamos todos os pacotes icmp da rede 192.168.0.0/16. Perceba que para o filtro "**net**" deve ser especificado somente os octetos referentes a rede, não há como especificar mascaras intermediárias além de 32/16/8/0.

Agora o *gran finale* onde iremos aprender como capturar pacotes filtrando por uma flag específica do cabeçalho TCP. Para isso precisamos de uma teoria básica sobre cabeçalho TCP. Vejamos a estrutura do cabeçalho TCP:

```
 0                           15                              31
-----------------------------------------------------------------
|          source port          |       destination port        |
-----------------------------------------------------------------
|                        sequence number                        |
-----------------------------------------------------------------
|                     acknowledgment number                     |
-----------------------------------------------------------------
|  HL   | rsvd  |C|E|U|A|P|R|S|F|        window size            |
-----------------------------------------------------------------
|         TCP checksum          |       urgent pointer          |
-----------------------------------------------------------------
```

Começando pelo octeto 0 o octeto que nos interessa é o 13 onde estão nossas famosas flags de controle.

```
 0             7|             15|             23|             31
----------------|---------------|---------------|----------------
|  HL   | rsvd  |C|E|U|A|P|R|S|F|        window size            |
----------------|---------------|---------------|----------------
|                |  13th octet   |               |               |
```

Vejamos este octeto mais de perto:

```
|C|E|U|A|P|R|S|F|
|---------------|
|7 6 5 4 3 2 1 0|
```

Essas são as famosas flags TCP CWR, ECE, URG, ACK, PSH, RST, SYN e FIN apresentadas exatamente nesta ordem. Perceba que a significância dos bits cresce da direita para a esquerda. Agora que já sabemos onde as flags estão digamos que queremos pacotes somente com a flag SYN ativada.

```
|C|E|U|A|P|R|S|F|
|---------------|
|0 0 0 0 0 0 1 0|
|---------------|
|7 6 5 4 3 2 1 0|
```

Traduzindo em binário nosso octeto 13 deve assumir o valor 00000010. Convertendo o mesmo para decimal:

```
   7     6     5     4     3     2     1     0
0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 1*2 + 0*2  =  2
```

Temos o número 2 em decimal. Agora como podemos falar para o tcpdump capturar somente os pacotes TCP que possuírem o octeto 13 com o valor decimal 2? Fácil, basta usar o filtro "**tcp[13] == 2**". Vejamos no exemplo abaixo:

```
smith ~ # tcpdump -i eth0 -n tcp[13] == 2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:46:10.978953 IP 192.168.0.2.34670 > 209.85.133.18.443: S 1113104022:1113104022(0) win 5840 <mss 1460,sackOK,timestamp 976173 0,nop,wscale 7>
11:46:20.362403 IP 192.168.0.2.47876 > 209.85.193.104.23: S 1282044372:1282044372(0) win 5840 <mss 1460,sackOK,timestamp 978519 0,nop,wscale 7>
11:46:25.040800 IP 192.168.0.2.54605 > 192.168.0.1.23: S 1360875102:1360875102(0) win 5840 <mss 1460,sackOK,timestamp 979688 0,nop,wscale 7>
```

Neste exemplo capturamos todos os pacotes TCP com a flag SYN ativada (igual à 1), porém necessariamente as outras flags devem estar desativadas (igual à 0). Mas eu quero todos os pacotes com a flag SYN ativada independente do estado das outras flags, tem como fazer isso? O tcpdump só não faz café. Vamos exemplificar para ficar mais fácil. Temos um pacote do tipo SYN-ACK como segue abaixo:

```
|C|E|U|A|P|R|S|F|
|---------------|
|0 0 0 1 0 0 1 0|
|---------------|
|7 6 5 4 3 2 1 0|
```

O filtro "**tcp[13] == 2**" não irá capturar o pacote pois neste caso o valor binário do octeto 13 é 00010010 (18 em decimal) e não 00000010 (2 em decimal). Agora que vem o pulo gato! Para que o tcpdump "*olhe*" somente para a flag SYN e descarte as demais você deve pedir a ele que faça um AND com o valor binário referente a flag SYN. Vejamos graficamente:

```
     00010010 (octeto 13 com as flags SYN-ACK)
AND  00000010 (valor binário referente a flag SYN)
     --------
=    00000010 (resultado do AND)
```

Perceba mesmo que todas as flags estivessem ativadas (1) ou desativadas (0) o resultado sempre seria 00000010. Traduzindo na língua do tcpdump o filtro ficaria da seguinte forma "**tcp[13] & 2 == 2**". Em outras palavras, você está dizendo o seguinte para o tcpdump: Pegue todos os pacotes TCP, faça um AND entre o valor decimal 2 com o valor do octeto 13 e me mostre somente os que resultarem o valor decimal 2.

Você pode utilizar está técnica para filtrar baseado em qualquer bit do cabeçalho TCP. Isso é extremamente útil em um trobleshooting onde se sabe exatamente os pacotes que se deseja capturar.

3. Conclusão

O tcpdump é uma ferramenta indispensável para um trobleshooting de rede. Como podemos ver ele é preciso como um bisturi onde podemos capturar exatamente o que pretendemos. O que vimos nesta documentação pode ser muito mais aproveitado com a leitura da man page do tcpdump em http://www.tcpdump.org/tcpdump_man.html.

4. Referências

- http://www.tcpdump.org
- http://www.ietf.org/rfc/rfc0791.txt
- http://www.ietf.org/rfc/rfc0793.txt
- http://www.google.com.br

Dúvidas, críticas e sugestões devem ser enviadas para ricardo.iramar@gmail.com.