

Lire des paquets capturés sur le réseau en Python

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 Décembre 2008

<http://www.bortzmeyer.org/libpcap-python.html>

Il y a des tas de raisons de vouloir lire soi-même les paquets capturés sur le réseau. Cela peut être par curiosité, par souci de sécurité ou bien pour faire des statistiques. Pour un rapide coup d'œil, on utilise en général tcpdump, pour un examen interactif détaillé, l'excellent Wireshark mais, pour des études à long terme, il vaut mieux programmer et développer un programme d'analyse spécifique.

Souvent les articles sur la question parlent ensemble de la **capture** des paquets et de leur **analyse**. Ici, je ne mentionne que l'analyse, supposant que la capture a été faite par d'autres moyens (<http://www.bortzmeyer.org/capture-paquets.html>), par exemple tcpdump-wFICHER ou bien pcapdump (<http://packages.debian.org/sid/pcaputils>).

L'outil d'analyse le plus fréquent pour les programmeurs C est libpcap (dont je parle dans « Lire des paquets capturés sur le réseau en C (<http://www.bortzmeyer.org/libpcap-c.html>) »). Et en Python? Il existe plusieurs interfaces à la libpcap, notamment pylibpcap (<http://pylibpcap.sourceforge.net/>) et pcap (http://oss.coresecurity.com/projects/pcapy.html). pcap semble de plus haut niveau et c'est celle que j'utilise.

pcapy peut faire la capture et la lecture des fichiers au format pcap. Ses capacités d'analyse sont limitées, on utilise en général la bibliothèque impacket (<http://oss.coresecurity.com/projects/impacket.html>), des mêmes auteurs, ou bien carrément scapy.

Voici un programme simple de lecture d'un fichier pcap, tel que produit par tcpdump :

```
import pcap

reader = pcap.open_offline("mydata.pcap")

while True:
    try:
        (header, payload) = reader.next()
        print "Got a packet of length %d" % header.getlen()
    except pcap.PcapError:
        break
```

Il ne semble pas y avoir de moyen de détecter la fin du fichier à part en récupérant l'erreur très générique `PcapError`.

`payload` est juste une chaîne non décodée d'octets, représentés en caractères. Si je veux l'analyser, je peux le faire « à la main » avec les fonctions de manipulation binaire de Python. Cela nécessite évidemment de lire les différentes normes pour connaître le format des paquets :

```
import pcap

reader = pcap.open_offline("mydata.pcap")

while True:
    try:
        (header, payload) = reader.next()
        # Ethernet type is two-bytes long and starts at 12 (before, you
        # have the MAC addresses)
        if payload[12:14] == '\x86\xDD': # 0x86DD is the Ethernet type for IPv6
            ip_payload = payload[14:]
            ip_version = (ord(ip_payload[0]) & 0xf0) >> 4
            if (ip_version != 6):
                raise Exception("Packet has ethernet type of IPv6 but internal IP version is %d" % ip_version)
            next_header = ord(ip_payload[6]) # RFC 2460, section 3 to know the offset
            print "Next header (probably the transport protocol) is %d" % next_header
    except pcap.PcapError:
        break
```

Ici, on teste le type indiqué dans l'en-tête (<http://www.iana.org/assignments/ethernet-numbers>) Ethernet pour ne garder que 0x86DD (IPv6). On saute les quatorze premiers octets (la taille de l'en-tête Ethernet) pour récupérer le paquet IPv6 que l'on decode en suivant le RFC 2460¹. Le numéro de version fait moins d'un octet, d'où les manipulations de bits avec `&` et `>>`.

Mais il peut être plus simple d'utiliser la bibliothèque `impacket` (<http://oss.coresecurity.com/projects/impacket.html>), conçue pour faciliter le décodage :

```
import pcap
import impacket.ImpactDecoder as Decoders

reader = pcap.open_offline("mydata.pcap")
decoder = Decoders.EthDecoder()

while True:
    try:
        (header, payload) = reader.next()
        result = decoder.decode(payload)
        # http://www.iana.org/assignments/ethernet-numbers
        if result.get_ether_type() == 0x86DD: # IPv6
            print "Got an IPv6 packet of length %d" % header.getlen()
    except pcap.PcapError:
        break
```

Ici, plus besoin de connaître la structure de l'en-tête Ethernet, on peut utiliser `get_ether_type()` au lieu de `payload[12:14]`.

Cela va permettre de s'attaquer à des protocoles de plus haut niveau, souvent plus complexes à decoder. Commençons par UDP :

1. Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc2460.txt>

```
import pcap
import impacket.ImpactDecoder as Decoders
import impacket.ImpactPacket as Packets

reader = pcap.open_offline("mydata.pcap")
eth_decoder = Decoders.EthDecoder()
ip_decoder = Decoders.IPDecoder()
udp_decoder = Decoders.UDPDecoder()

while True:
    try:
        (header, payload) = reader.next()
        ethernet = eth_decoder.decode(payload)
        if ethernet.get_ether_type() == Packets.IP.ethertype:
            ip = ip_decoder.decode(payload[ethernet.get_header_size():])
            if ip.get_ip_p() == Packets.UDP.protocol:
                udp = udp_decoder.decode(
                    payload[ethernet.get_header_size()+ip.get_header_size():])
                print "IPv4 UDP packet %s:%d->%s:%d" % (ip.get_ip_src(),
                                                         udp.get_uh_sport(),
                                                         ip.get_ip_dst(),
                                                         udp.get_uh_dport())

    except pcap.PcapError:
        break
```

Ce programme affiche les paquets UDP sur IPv4 en indiquant les adresses et ports de source et de destination. On note qu'il n'y a pas eu besoin de lire le RFC 768, tout est pris en charge par Impacket (y compris le problème, que j'ai soigneusement évité dans l'exemple du décodage manuel, de l'ordre des octets dans les champs multi-octets). Il faut juste penser, à chaque fois qu'on grimpe d'une couche, à se décaler dans le paquet (du résultat de `get_header_size()` sur la couche précédente).

Pour le décodage de l'en-tête IP, Impacket ne dispose malheureusement que d'un décodeur IPv4 (<http://stackoverflow.com/questions/369764/ipv6-decoder-for-pcapimpacket>). Pour IPv6, il faut l'écrire soi-même, peut-être plutôt en utilisant Scapy (<http://www.secdev.org/projects/scapy/>). Ce sera pour un autre article.