Écoute du réseau et usurpation d'identité

(*"Les aventures de Sniff et Spoof . . . "*)

Rappels sur les sockets de transport
Les raw-sockets de réseau
Les packet-sockets de liaison
Quelques manœuvres "osées" . . .

Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

Propos

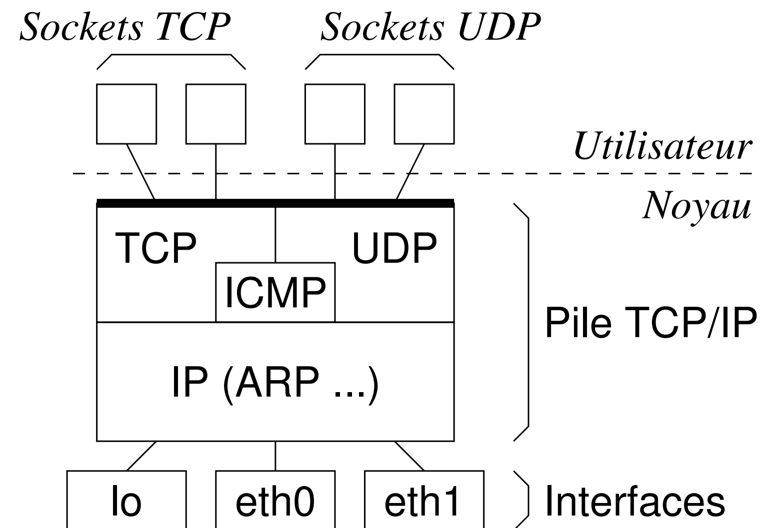
- *"Sniff"* **: écouter le trafic du réseau**
  - En particulier les informations qui ne nous sont pas destinées
- *"Spoof"* **: se faire passer pour une autre machine**
  - Émettre *"discrètement"* des informations falsifiées
  - Influencer le comportement des autres machines
    - En particulier pour pouvoir mieux écouter

Les *sockets* de niveau transport

▷ **Dédiées aux applications**

◇ Attachées à une de nos adresses *IP* (ou `INADDR_ANY`) et un port

◇ Pas de privilèges requis (sauf ports<1024)



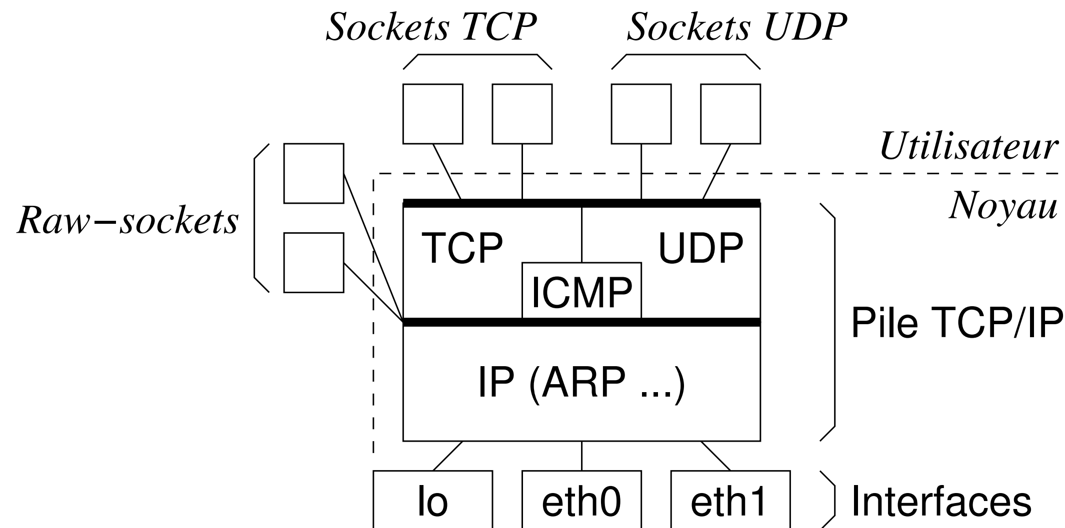
Les *sockets* de niveau transport

- **Les entêtes des *segments/datagrammes* sont implicites**
  - On ne peut pas les rédiger, on ne fournit que les données
  - Les entêtes nous identifient formellement
    → On ne peut pas rester *“discret”* !!!
- **On ne reçoit que ce qui nous concerne**
  - Les *segments/datagrammes* nous sont explicitement adressés
    → On ne peut pas espionner le reste du trafic !!!

Les *raw-sockets* de niveau réseau

▷ **Dédiées à la mise au point de protocoles de transport**

- ◇ Attachées à une de nos adresses *IP* (ou `INADDR_ANY`) (port=0)
- ◇ Permettent d'envoyer/recevoir des *datagrammes IP*
- ◇ Des privilèges sont requis
- ◇ `man 7 socket`, `man 7 ip`, `man 7 raw`



Les *raw-sockets* de niveau réseau

```
//---- Create socket ----
int on;
int fd=socket(PF_INET,SOCK_RAW,IPPROTO_RAW); // IPPROTO_ICMP, IPPROTO_UDP, IPPROTO_TCP ...
if(fd==-1)
  {
  perror("socket");
  return -1;
  }

//---- Provide IP header when writing ----
on=1;
if(setsockopt(fd,SOL_IP,IP_HDRINCL,&on,sizeof(int))==-1)
  {
  perror("setsockopt IP_HDRINCL");
  close(fd);
  return -1;
  }

//---- Allow broadcast ----
on=1;
if(setsockopt(fd,SOL_SOCKET,SO_BROADCAST,&on,sizeof(int))==-1)
  {
  perror("setsockopt SO_BROADCAST");
  close(fd);
  return -1;
  }
```

Les *raw-sockets* de niveau réseau

▷ **L'appel à `socket()`**

- ◇ Domaine `PF_INET` → ça reste de l'*IP* !
- ◇ Type `SOCK_RAW`
  → pas uniquement *UDP* (`SOCK_DGRAM`) ou *TCP* (`SOCK_STREAM`)
- ◇ Protocole à filtrer
  - ∘ `IPPROTO_ICMP`, `IPPROTO_UDP`, `IPPROTO_TCP` . . .
    → réception uniquement de celui-ci
  - ∘ `IPPROTO_RAW` → Aucune réception possible !!!
    - ∘ On émet ce qu'on veut (`IP_HDRINCL` obligatoire)
  - ∘ Un numéro de protocole inutilisé
    - ∘ Voir `/usr/include/netinet/in.h` ou `/etc/protocols`
    - ∘ Permet la mise au point d'un nouveau protocole de transport

Les *raw-sockets* de niveau réseau

- **L'appel à `setsockopt(IP_HDRINCL)`**
  - L'entête *IP* **doit** être fourni lors de l'émission (`on=1`)
  - Champs *"longueur totale"* et *"checksum"*
    - Toujours remplis automatiquement
  - Champs *"adresse source"* et *"identification"*
    - Remplis automatiquement si nuls
  - Si non utilisé (`on=0`) l'entête *IP* est implicite
    - Utiliser `setsockopt()` pour le paramétrer
- **Choix du protocole et `IP_HDRINCL`**
  - On peut écouter un protocole et émettre plusieurs
  - Ex : échanges *UDP* + erreurs *ICMP*

Les *raw-sockets* de niveau réseau

▷ **Intercalées entre *IP* et *TCP*, *UDP*, *ICMP* ...**

◇ On peut se faire passer pour un service de notre machine

◇ On peut écouter ce qui s'adresse aux services de notre machine

∘ Les *datagrammes* reçus atteignent leurs services de destination

▷ **Les entêtes des *datagrammes* sont en partie implicites**

◇ Même avec `IP_HDRINCL` la source est implicite

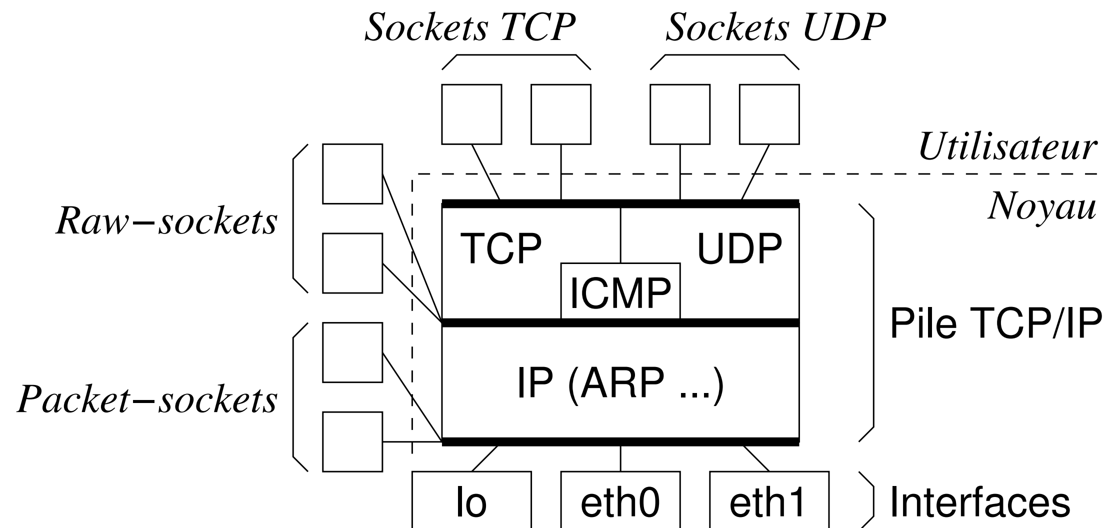
◇ Les entêtes nous identifient formellement
→ On ne peut pas rester *"discret"* !!!

▷ **On ne reçoit que ce qui nous concerne**

◇ Les *segments/datagrammes* nous sont explicitement adressés
→ On ne peut pas espionner le reste du trafic !!!

Les *packet-sockets* de niveau liaison

- **Dédiées à la mise au point de protocoles de réseau**
  - Attachées à une de nos interfaces
  - Permettent d'envoyer/recevoir des *trames MAC*
  - Des privilèges sont requis
  - `man 7 packet`



Les *packet-sockets* de niveau liaison

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

#include <netinet/ether.h>
#include <netinet/in.h>
#include <net/if.h>
#include <netpacket/packet.h>

const char * name="eth0";

//---- Create socket ----
int fd=socket(PF_PACKET,SOCK_RAW,htons(ETH_P_ALL));
if(fd==-1)
  { perror("socket"); return -1; }

//---- Find interface index ----
struct ifreq ifr;
memset(&ifr,0,sizeof(struct ifreq));
strncpy(ifr.ifr_name,name,sizeof(ifr.ifr_name));
if(ioctl(fd,SIOCGIFINDEX,&ifr)==-1)
  { perror("ioctl SIOCGIFINDEX"); close(fd); return -1; }
int index=ifr.ifr_ifindex;
```

Les *packet-sockets* de niveau liaison

```
//---- Bind socket to interface ----
struct sockaddr_ll addr;
memset(&addr,0,sizeof(struct sockaddr_ll));
addr.sll_family=AF_PACKET;
addr.sll_ifindex=index;
addr.sll_protocol=htons(ETH_P_ALL);
if(bind(fd,(struct sockaddr *)&addr,sizeof(struct sockaddr_ll))==-1)
  { perror("bind"); close(fd); return -1; }

//---- Determine MTU ----
memset(&ifr,0,sizeof(struct ifreq));
strncpy(ifr.ifr_name,name,sizeof(ifr.ifr_name));
if(ioctl(fd,SIOCGIFMTU,&ifr)==-1)
  { perror("ioctl SIOCGIFMTU"); close(fd); return -1; }
unsigned int mtu=ifr.ifr_mtu;

//---- Switch to promiscuous mode ----
struct packet_mreq mr;
memset(&mr,0,sizeof(struct packet_mreq));
mr.mr_ifindex=index;
mr.mr_type=PACKET_MR_PROMISC;
if(setsockopt(fd,SOL_PACKET,PACKET_ADD_MEMBERSHIP,
              &mr,sizeof(struct packet_mreq))==-1)
  { perror("setsockopt PACKET_MR_PROMISC"); close(fd); return -1; }
```

Les *packet-sockets* de niveau liaison

- **L'appel à `socket()`**
  - Domaine `PF_PACKET` → niveau liaison
  - Type `SOCK_RAW` → *trames* brutes avec entête de liaison (`SOCK_DGRAM` : entête retiré)
  - Protocole à filtrer
    - `ETH_P_IP`, `ETH_P_ARP` ... → réception uniquement de celui-ci
    - `ETH_P_ALL` → aucun filtrage sur le protocole, tout recevoir
    - On peut écrire ce qu'on veut !!!
- **L'appel à `bind()`**
  - Attachement à une interface désignée par un index
  - Index déterminé à partir du nom (`ioctl(SIOCGIFINDEX)`)
  - Si non attachée, réception depuis toutes les interfaces
    - Émission impossible

Les *packet-sockets* de niveau liaison

- **Détermination de la *MTU***
  - Déterminée à partir du nom de l'interface (`ioctl(SIOCGIFMTU)`)
  - Permet de dimensionner les tampons d'émission/réception
    - Utilisation de `send()`/`recv()` ou `read()`/`write()`
- **Passage en mode *promiscuous***
  - Éliminer le filtrage selon l'adresse *MAC* destination
    - Les *trames* qui ne concernent pas notre machine sont reçues
      → Possibilité d'écouter tout le trafic !!!
  - C'est une propriété de la *socket*
    - Propriété de l'interface dans les anciennes implémentations

Les *packet-sockets* de niveau liaison

- ▷ **Intercalées entre les interfaces et *IP***
  - ◇ On peut écouter ce qui concerne notre machine
    - ∘ Les *trames* reçues atteignent leur objectif *ARP*, *IP* …
- ▷ **Les entêtes des *trames* sont explicites**
  - ◇ On rédige nous même les adresses *MAC* source et destination
    → On peut rester *“discret”* !!!
- ▷ **En mode *promiscuous* on reçoit tout ce qui passe**
  - ◇ Les *trames* ne nous sont pas explicitement adressées
    → On peut espionner tout le trafic !!!
  - ◇ Limité par le cloisonnement (*switch*, sous-réseaux …)
- ▷ **Rédiger et interpréter les *trames***
  - ◇ Structures de données représentant les entêtes
  - ◇ Implémenter les *cheksums*, la fragmentation …

Écouter le trafic du réseau local

▷ **Utiliser une *packet-socket* en mode *promiscuous***

- ◇ Écoute passive, on n'émet rien
- ◇ Interpréter le contenu des trames
  - ◦ Reconstituer les séquences de segments *TCP*
  - ◦ Pages visitées, *e-mails* reçus/expédiés, mots de passe . . .
- ◇ Des outils effectuent cette opération :
  - ◦ *tcpdump* : affiche dans la console les *trames* capturées
  - ◦ *snort* : détecteur d'intrusion (bibliothèque d'attaques connues)
  - ◦ . . .

Écouter le trafic du réseau local

- **Différence entre** *hub* **et** *switch*
  - Le concentrateur (*hub*) fonctionne au niveau 1 *OSI* (physique)
    - Le signal se propage sur tous les brins
    - Une écoute passive suffit à recueillir les trames de tous les nœuds
  - Le commutateur (*switch*) fonctionne au niveau 2 *OSI* (liaison)
    - Une trame n'est émise que vers le brin relié au destinataire
    - On ne reçoit que les trames qui nous sont explicitement destinées (notre adresse *MAC* ou celle de diffusion)
    - Ceci donne une **fausse** impression de sécurité !
    - L'emploi d'un procédé actif permet d'écouter *"derrière"* un *switch*

Écouter le trafic du réseau local

▷ **Écouter derrière un** *switch*

- ◇ Seul le trafic qui nous concerne nous atteint !!!
  → envoyer des *trames* pour détourner le trafic vers notre machine
- ◇ La solution de *"bourrin"* pour *switch* *"bon marché"*
  - ∘ Envoyer au *switch* un déluge de *trames* (plusieurs machines ?)
  - ∘ S'il ne peut plus traiter toutes les *trames* il se comporte en *hub* !!!
- ◇ Une solution plus discrète : *ARP-spoofing* (*ARP-cache-poisoning*)

Écouter le trafic du réseau local

- **Écouter derrière un** *switch* **par** *ARP-spoofing*
  - Modifier le cache *ARP* de la victime
    - L'adresse *IP* de la cible doit correspondre à une fausse adresse *MAC*
    - Pour atteindre la cible, la victime utilisera cette fausse adresse
    - Le *switch* acheminera légitimement les trames vers cette fausse adresse
    - Il suffit d'écouter en mode *promiscuous* (`tcpdump -i eth1 -nqAs0 ...`)
  - Ne pas attendre qu'un nœud fasse une requête *ARP*
    - La *"vraie"* réponse risque d'annuler la notre
  - En théorie, il suffit d'envoyer une réponse (*RFC-0826*)
    - En pratique, seules les requêtes provoquent une insertion sûre (cf le cours *"Fonctionnement de base des réseaux"*)

Écouter le trafic du réseau local

- **Écouter derrière un** *switch* **par** *ARP-spoofing*
  - On émet dans une trame `FakeMAC`→`VictimMAC` la requête *ARP*
    `[FakeMAC, TargetIp, 00:00:00:00:00:00, VictimIP]`
  - La source est `FakeMAC` → le *switch* la localise sur notre brin
  - La victime insère dans son cache *ARP* la paire `TargetIP`/`FakeMAC`
  - Elle émet alors dans une trame `VictimMAC`→`FakeMAC` la réponse *ARP*
    `[VictimMAC, VictimIP, FakeMAC, TargetIP]`
  - On ignore simplement cette réponse
  - La victime utilisera dorénavant `FakeMAC` pour joindre `TargetIP`
  - Renouveler l'opération de temps en temps pour éviter l'expiration du cache
  - nb : On pourrait utiliser notre propre adresse *MAC* mais c'est moins discret (ici `FakeMAC` est une adresse inutilisée dans le sous-réseau)

Écouter le trafic du réseau local

- **Écouter derrière un** *switch* **par** *ARP-spoofing*
  - Entre un client et un serveur dans le même sous-réseau
    - Se faire passer pour le client auprès du serveur et réciproquement
    - Relayer les trames en utilisant les bonnes adresses *MAC*
      (les échanges client↔serveur doivent avoir lieu normalement)
  - Si le serveur est dans un autre sous-réseau
    - Même démarche mais entre le client et le routeur
  - Entre `N` clients potentiels et un serveur (ou un routeur)
    - Se faire passer pour le serveur (ou le routeur) auprès de chaque client
    - Lorsqu'un client tente d'accéder au serveur
      → Se faire passer pour le client auprès du serveur
      → Évite le débordement du cache *ARP* du serveur (ou routeur)

Quelques outils

- **Bibliothèques**
  - *libpcap* : capture de *trames* (`http://www.tcpdump.org/`)
  - *libnids* : réassemblage *IP*, suivi *TCP* (`http://libnids.sourceforge.net/`)
  - *libnet* : rédaction et envoi de *segments*, *datagrammes*, *trames* ... (`http://www.packetfactory.net/Projects/Libnet/`)
- **Utilitaires**
  - *tcpdump* : afficher dans la console les *trames* capturées (`http://www.tcpdump.org/`)
  - *wireshark* : interface graphique pour capturer et analyser les sessions (`http://www.wireshark.org/`)
  - *snort* : détecteur d'intrusion (bibliothèque d'attaques connues) (`http://www.snort.org/`)
  - *nmap* : découverte de réseaux, d'*OS*, de services (`http://www.insecure.org/nmap/`)