TP : Créer un Firewall avec eBPF et XDP

Table des matières

Objectifs pédagogiques

À la fin de ce TP, vous serez capable de :

- Installer BCC/XDP et vérifier le support
- Comprendre la structure Ethernet -> IP -> transport
- Filtrer les paquets par protocole, IP ou port
- Charger des règles dans les maps BPF depuis Python
- Composer un mini-firewall multi-critères

Partie 0 : Installation

0.1 Installation

1. `sudo apt-get update`
2. `sudo apt-get install -y python3-bpfcc bpfcc-tools linux-headers-$(uname -r)`
3. `python3 -c "from bcc import BPF; print('BCC OK')"`

0.2 Support XDP

- Kernel >= 4.8
- `sudo bpftool feature probe | grep xdp`
- `ip link show` pour trouver une interface physique

Note : XDP ne fonctionne pas forcément sur une interface virtuelle ou sur `lo`.

Partie 1 : DROP global

Objectif

Bloquer tous les paquets entrants avec XDP.

Code source : `ex1_xdp_drop_all.py` contient l'eBPF `xdp_drop_all`, le chargement, la liaison, le compteur et la boucle d'affichage. Complétez les sections marquées « À COMPLÉTER ».

Questions

- Pourquoi `__sync_fetch_and_add()` plutôt que `count++` ?
- Quel impact sur votre connexion lors du DROP ?
- Pourquoi `ping localhost` fonctionne encore ?

Tests

```
Terminal 1:
sudo python3 ex1_xdp_drop_all.py MA_CARTE_RESEAU
Terminal 2:
ping 8.8.8.8 (devra échouer)
```

Partie 2 : Filtrage par protocole (ICMP)

Objectif
Bloquer uniquement les paquets ICMP.

Code source : `ex2_xdp_block_icmp.py` parse Ethernet + IP, compare `ip->protocol`, remplit `stats` et retourne `XDP_DROP` pour ICMP.

Questions
- Pourquoi vérifier `(void *)(eth + 1) > data_end` ?
- Rôle de `htons(ETH_P_IP)` ?
- Que faire des paquets IPv6 / ICMPv6 ?

Partie 3 : Blocklist IP

Objectif
Bloquer une adresse IP source (ou destination en défi).

Code source : `ex3_xdp_block_ip.py` convertit l'IP fournie (`socket.inet_aton()` + `struct.unpack('I', ...)`), charge `xdp_block_ip` et place les compteurs `stats`.

Questions
- Différence `ip->saddr` vs `ip->daddr` ?
- Pourquoi l'affichage est-il différent de la mémoire ?
- Comment bloquer source et destination ?

Partie 4 : Filtrage par port TCP/UDP

Objectif
Bloquer un port destination (ex : 80).

Code source : `ex4_xdp_block_port.py` inspecte Ethernet/IP/TCP/UDP, récupère `ntohs(tcp->dest)` ou `ntohs(udp->dest)` et compare au port à bloquer.

Questions
- Pourquoi `ip->ihl * 4` ?
- Différence entre `tcp->dest` et `tcp->source` ?
- Qu'est-ce que `ntohs()` corrige ?

Partie 5 : Firewall piloté par configuration

Format `firewall.conf`
```
IP_LIST=1.1.1.1,8.8.8.8
PORT_LIST=80,443
```

Code source : `ex5_xdp_firewall_config.py` lit la config, remplit `blocked_ips` / `blocked_ports`, compile `xdp_firewall`, attache l'interface et affiche les statistiques.

Tests

```
Terminal 1:
sudo python3 ex5_xdp_firewall_config.py MA_CARTE_RESEAU firewall.conf
Terminal 2:
ping 1.1.1.1  (bloqué)
ping 9.9.9.9  (pass)
curl http://example.com  (bloqué sur le port 80)
curl https://example.com (passe)
```

Questions
- Pourquoi les maps BPF sont-elles préférables à un `if/else` ?
- Que se passe-t-il si vous modifiez `firewall.conf` en live ?

- Pourquoi séparer IPs et ports ?

Partie 6 : Projet final - Firewall complet

Créez `ex6_xdp_firewall_full.py` avec :

- Règles multi-critères (IP source/destination, port, protocole)
- Actions `ALLOW` / `DENY`
- Logging des paquets bloqués
- Interface CLI (add / list / stats)

Structure suggérée :

```
struct fw_rule {
  u32 src_ip;
  u32 dst_ip;
  u16 dst_port;
  u8 protocol;
  u8 action; // 0=PASS, 1=DROP
};
BPF_HASH(rules, u32, struct fw_rule, 256);
```

Usage

```
firewall> add any 1.1.1.1 0 any DROP
firewall> add any any 80 tcp DROP
firewall> list
firewall> stats
```

Consignes :

1. Reprenez la logique des exercices précédents
2. Combinez les différents types de filtrage
3. Testez chaque fonctionnalité séparément
4. Documentez votre code avec des commentaires

Annexes

Commandes utiles

```
sudo ip link set dev MA_CARTE_RESEAU xdp off
sudo bpftool prog list
sudo bpftool map list
sudo dmesg | tail -20
```

Debugging

- utilisez `BPF(text=bpf_text, debug=0x04)` pour voir le C compilé
- erreurs fréquentes : helper non autorisé, back-edge, operation not supported

Référence rapide

```
// Ethernet (14 bytes)
struct ethhdr {
    unsigned char h_dest[6];    // MAC destination
    unsigned char h_source[6];  // MAC source
    __be16 h_proto;             // ETH_P_IP, ETH_P_IPV6
};

// IP (20+ bytes, variable avec options)
struct iphdr {
    __u8 ihl:4;       // Header length (en mots de 32 bits)
    __u8 version:4;
    __u8 protocol;    // IPPROTO_TCP=6, UDP=17, ICMP=1
    __be32 saddr;     // IP source
```

```
    __be32 daddr;     // IP destination
};

// TCP (20+ bytes)
struct tcphdr {
    __be16 source;    // Port source
    __be16 dest;      // Port destination
};

// UDP (8 bytes)
struct udphdr {
    __be16 source;
    __be16 dest;
    __be16 len;
    __be16 check;
};
```

À rendre

1. **Code source**: tous vos scripts Python complétés (voir les `ex*_*.py` mentionnés dans chaque exercice).
2. **Rapport PDF**: contenant :
   - Captures des tests pour chaque exercice
   - Réponses aux questions de réflexion
   - Difficultés rencontrées et solutions
   - Analyse des résultats et des compromis choisis
3. **README.txt**: décrivant comment exécuter chaque script (ex : `sudo python3 ex1_xdp_drop_all.py <interface>`).

Format attendu : archive `nom_prenom_tp_xdp.zip`.

Conseils finaux

1. Testez chaque exercice graduellement, ne passez pas au suivant tant que le précédent n'est pas stable.
2. Lisez attentivement les erreurs du compilateur eBPF ; elles référencent souvent la ligne fautive.
3. Utilisez `tcpdump` (`sudo tcpdump -i MA_CARTE_RESEAU -c 10`) pour vérifier que les paquets atteignent l'interface.
4. Commentez votre code pour faciliter la révision future (et votre propre relecture).
5. Lorsque vous demandez de l'aide, indiquez ce que vous avez déjà essayé et quelles ressources vous avez consultées.

Ce TP est conçu pour vous faire apprendre en pratiquant ; ce n'est pas le code final qui compte le plus, mais la compréhension que vous en aurez.