Programmation et attaques réseaux sous Linux

Developpement d'applications sécurisées

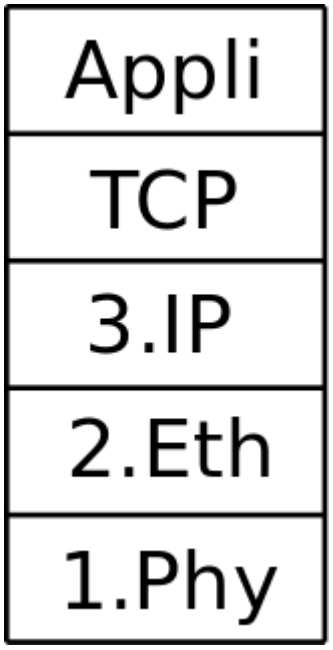
Cours/TD niveau avancé.

Nicolas Gama

Les couches OSI

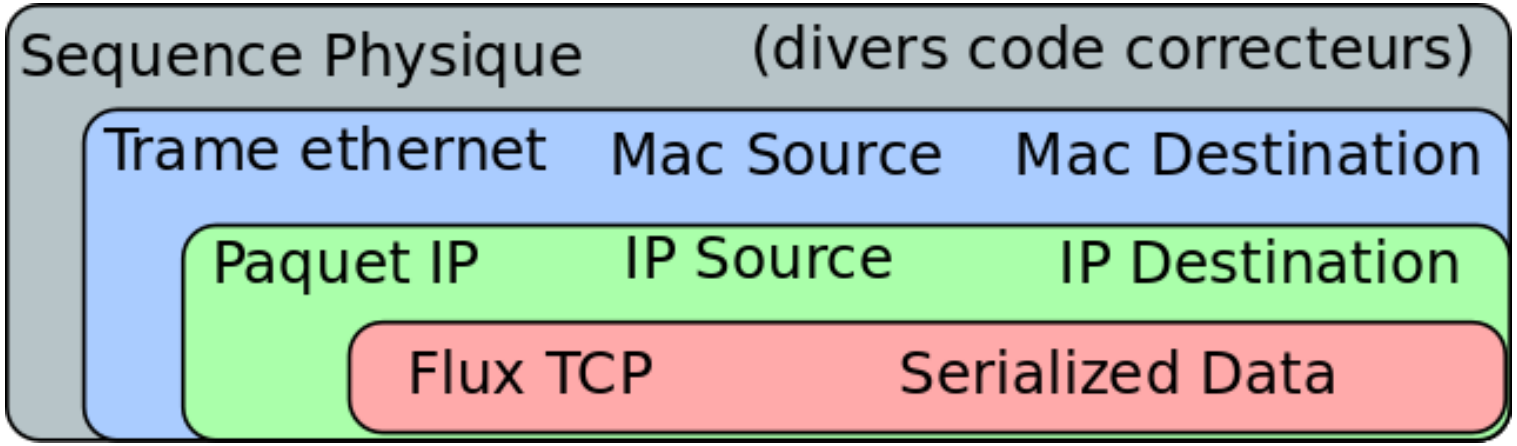
- Selon les modèles (OSI, TCP, ...), la gestion des packets réseaux est organisée en couches.
- La couche supérieure contiennent la représentation sémantique des objets
- La couche inférieure représente les 0 et les 1 qui transitent physiquement dans le cable réseau.
- (vous le saviez déjà!)

Les couches (presque OSI)



- **Applicatif:**
  Des objets haut niveau et intelligibles
- **TCP:**
  une représentation sérialisée (et/ou encodée) des objets applicatifs
- **IP:**
  Des paquets avec une IP emetteur et une IP destinataire.
  La portée des IP est potentiellement mondiale
- **Ethernet:**
  Des trames avec une adresse mac de source/destination.
  La portée des adresses mac est limitée au réseau switché.
- **Physique:**
  Des 0 et des 1 avec des codes correcteurs, qui transitent dans le cable réseau.

Encapsulation

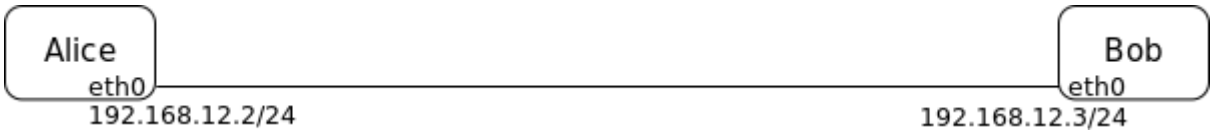


- Les couches supérieures sont **encapsulées** dans les couches immédiatement inférieures.
- Il y a aussi de la **fragmentation** (pas important ici)

Grand principe de l'informatique

- Un programme informatique revient très souvent à:
  - Prendre une information à un endroit (source), sous un certain format
  - **Transférer l'information** à un autre endroit, sous un format légèrement différent.
  - Répéter cela en boucle, ou de manière hiérarchique

Client-Serveur TCP



- Bob lance un serveur sur le port 4444
- Alice se connecte (sur 192.168.12.3:4444)
- Les deux obtiennent une Socket TCP,
  qui leur permet d'échanger des octets

Programmation TCP en Java

```
=======================================
ALICE
=======================================


//alice se connecte
bSocket = new Socket(192.168.12.3,4444);

//récupérer les IO de la socket
bOut = bSocket.getOutputStream();
bIn  = bSocket.getInputStream();

//recevoir puis envoyer un byte
int x = bIn.read();
int y=456; bOut.write(y);
```

```
=======================================
BOB
=======================================
//créer un serveur
ss = new ServerSocket(4444);
//bob attend la connexion d'alice
aSocket = ss.accept();

//idem
aIn  = aSocket.getInputStream();
aOut = aSocket.getOutputStream();

//envoyer puis recevoir un byte
int x=123; aOut.write(x);
int y = aIn.read();
```

Sérialisation de données

- Toutes les données doivent etre sérialisées pour passer à travers la socket.
- Il existe des formats standards: JSON, XML
  - Très standard, interopérable.

- Java possède un mécanisme de sérialisation binaire, via les Object Streams.
  - Moins inter-opérable, mais quand même pratique.

Sérialisation java

```
====================================
ALICE
====================================
[…]
//récupérer les IO de la socket
bOut = bSocket.getOutputStream();
hOut = new ObjectOutputStream(bOut)

//envoyer un objet complexe
GrosObjetComplexe x = […];
hOut.writeObject(x);
```

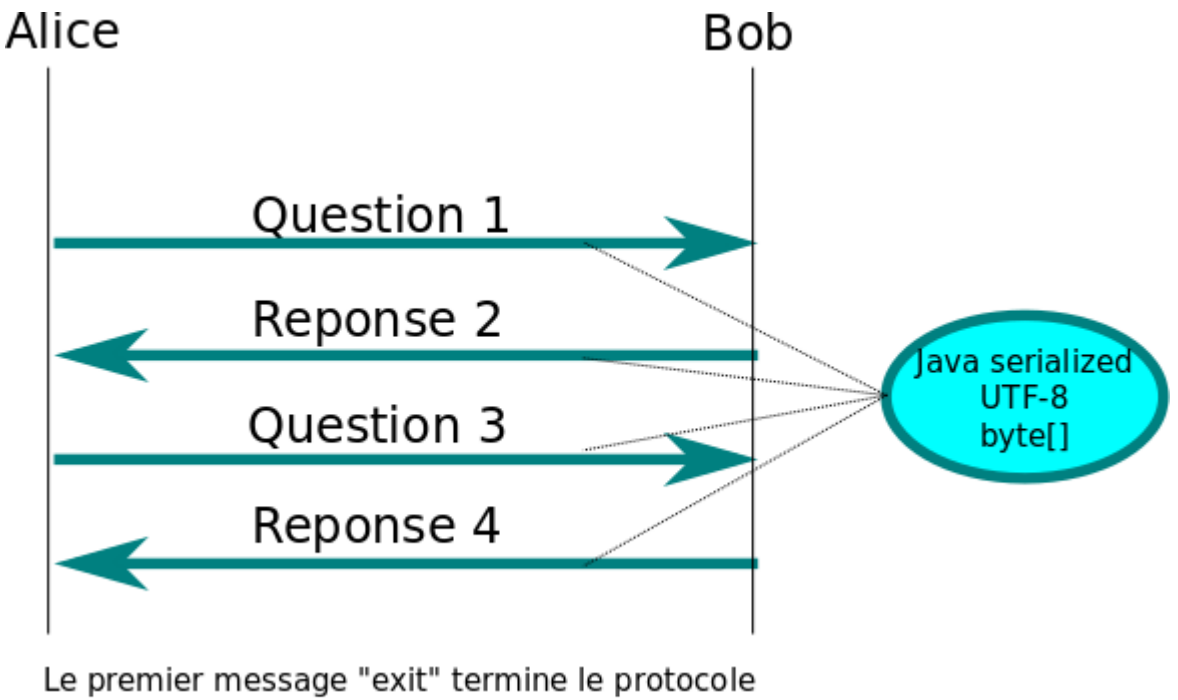
```
====================================
BOB
====================================
[…]
//idem
aIn = aSocket.getInputStream();
hIn = new ObjectInputStream(aIn)

//recevoir un objet complexe

x = (GrosObjetComplexe) hIn.readObject();
```

Applicatif: De manière générale

- L'application définit un protocole utilisateur (dialogue, échange d'entités)

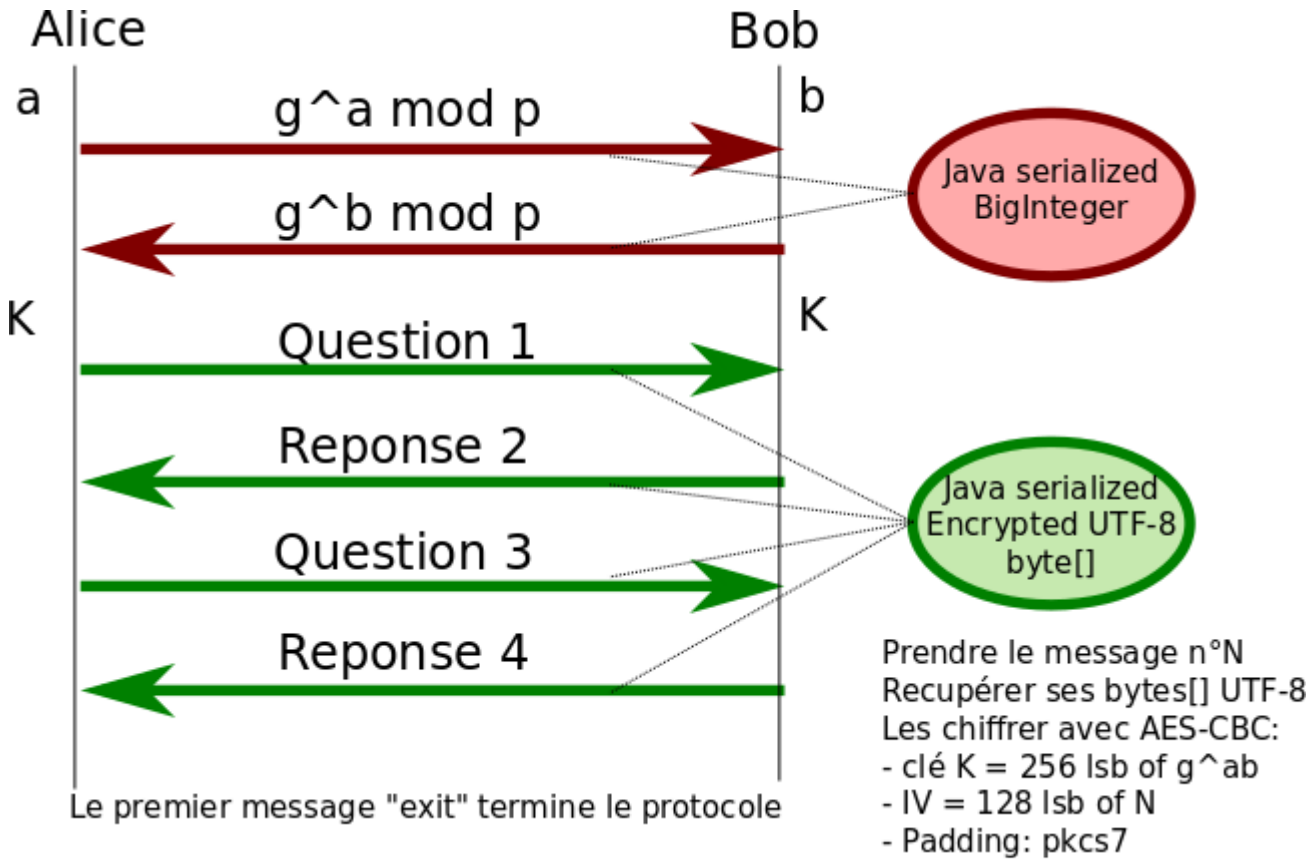
- L'application définit l'encodage, qui permet de sérialiser/encoder les entités
  - Cela peut utiliser des formats classiques
  - Cela peut aussi utiliser de la crypto!

Protocole applicatif simple

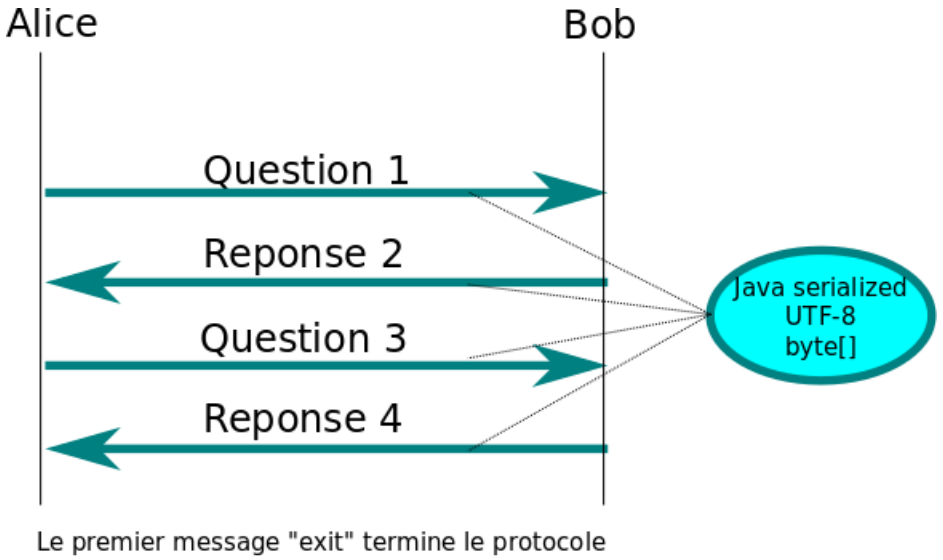


Le premier message "exit" termine le protocole

Protocole applicatif Diffie Hellmann

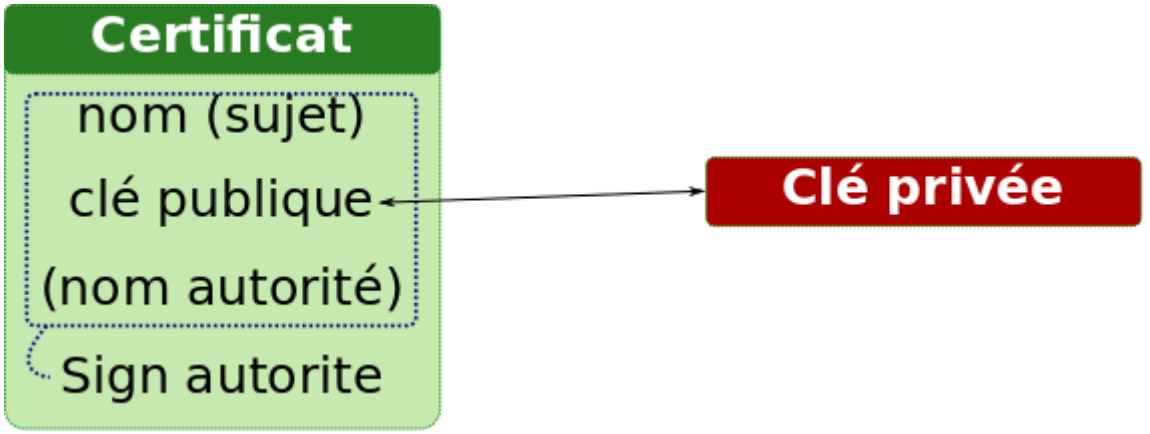


Protocole applicatif SSL/TLS



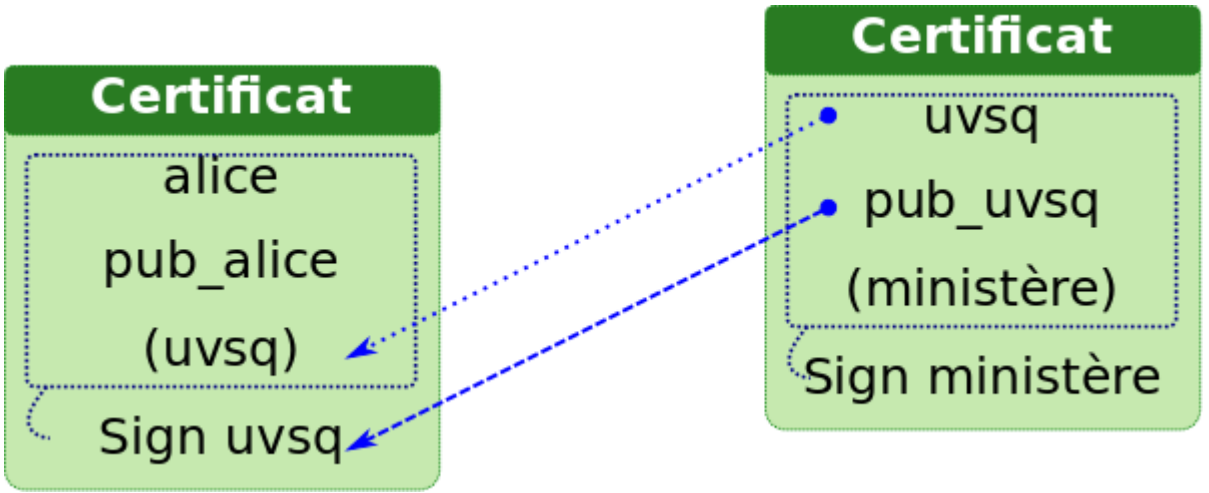
- On reprend exactement la version simple
- Sauf qu'on utilise une socket SSL au lieu d'une socket normale
  - C'est comme s'il y avait une couche supplémentaire entre l'applicatif et TCP (cf. vrai modèle OSI)

Certificats (rappels de base)



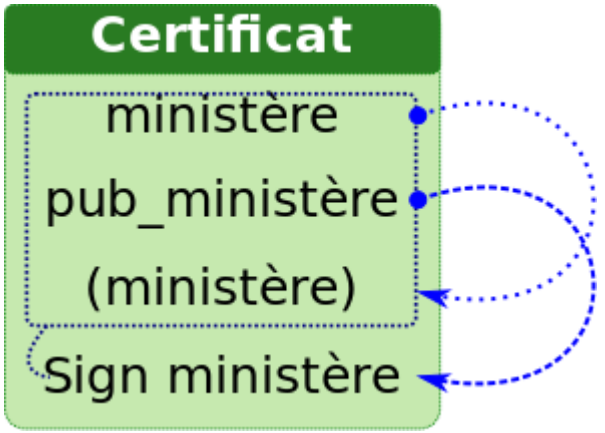
- Un certificat est une donnée publique
  Tout le monde peut en avoir une copie.
- Le **propriétaire** d'un certificat
  est celui qui détient sa clé privée
  - Lui seul peut déchiffrer des challenges
    chiffrés avec la clé publique (verification)

Certification



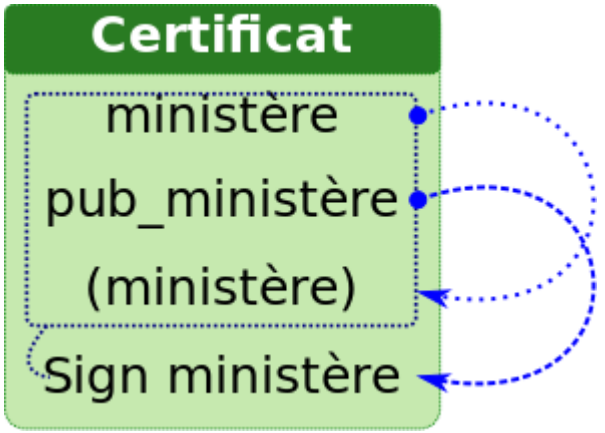
- L'émetteur (issuer) du certificat d'alice est le certificat de la clé publique qui a signé celui d'alice.
- Un certificat n'a qu'un seul emetteur.
- L'emetteur est publiquement vérifiable
  (ici, il suffit juste vérifier la signature de l'uvsq)

Certificat racine



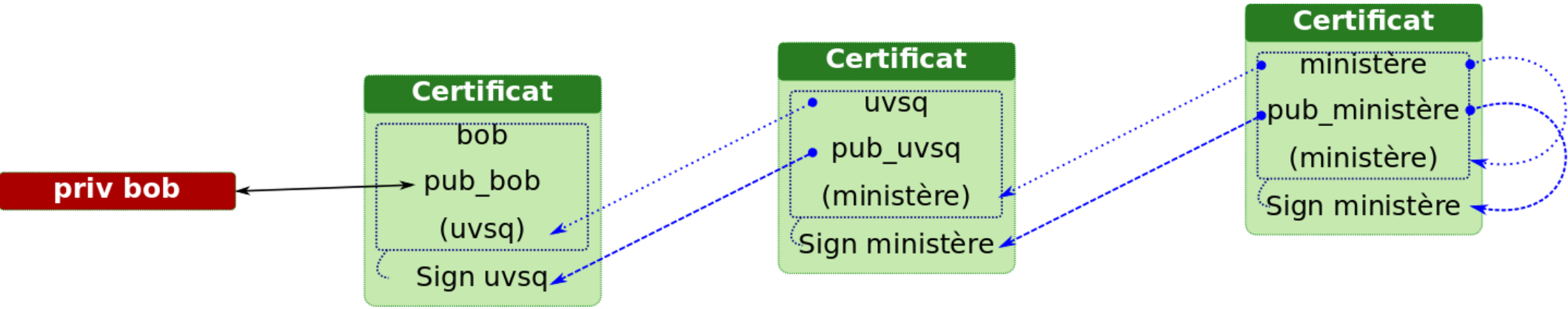
- Un certificat racine est émis par lui-même.
- Chaque utilisateur est libre de faire confiance ou non à un certificat racine.

Truststore



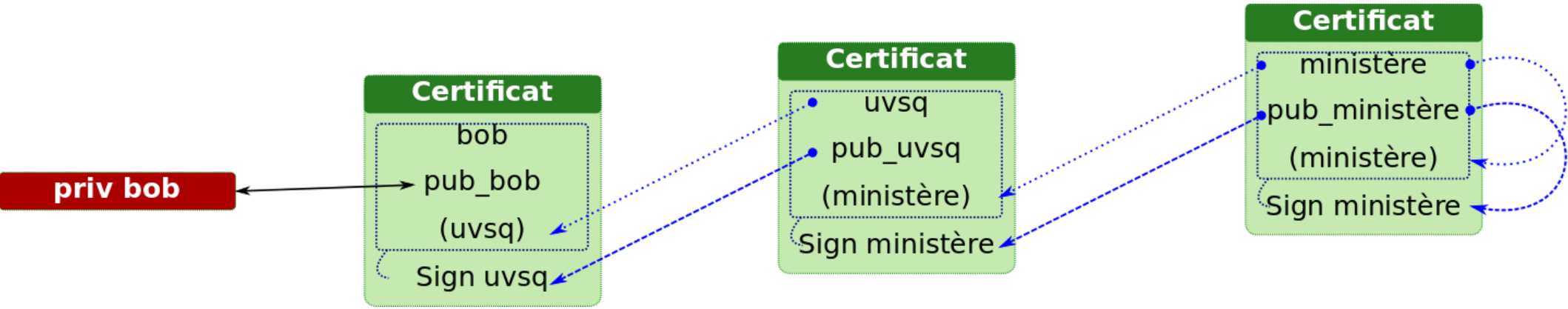
- Un truststore est une liste de certificats (en général racine) auquel l'utilisateur fait confiance.
- Ici, Alice choisit de mettre le certificat du ministère dans son truststore.
- **Note:** Le truststore par défaut de java (et de vos navigateurs web) contient environ 200 certificats https racine (Verisign, Thawte, Securom, ...).

Keystore (chaîne de certification)



- Pour s'authentifier par certificat, Bob doit posséder un keystore contenant toute la chaîne de certification jusqu'à la racine

Keystore (chaîne de certification)



- Le keystore de Bob sera validé par Alice si:
  - Bob est bien propriétaire du certificat initial (challenge de déchiffrement, la clé n'est pas révélée)
  - La chaine de certification est correcte.
  - Un des certificats de la chaîne est dans le truststore d'Alice

SSL/TLS

- Le serveur doit avoir un keystore
- Le client valide le serveur avec son truststore
- La clé publique du serveur permet d'établir une clé de session (AES en général)
- Tout le reste est chiffré avec la clé de session
- Toutes ces opérations sont faites **de manière transparente**:
  - le programmeur envoie des messages clairs dans la socket SSL comme si c'était une socket TCP classique!

Programmation SSL en Java

```
=========================================
ALICE
=========================================



//alice se connecte
scf = SSLSocketFactory.getDefault();
bSocket = scf.createSocket("host",4444);

//Tout le reste est pareil qu'avant
//récupérer les IO de la socket
bOut = bSocket.getOutputStream();
bIn  = bSocket.getInputStream();

//recevoir puis envoyer un byte
int x = bIn.read();
int y=456; bOut.write(y);
```

```
=========================================
BOB
=========================================
//créer un serveur
ssf = SSLServerSocketFactory
          .getDefault();
ss = ssf.createServerSocket(4444);
//bob attend la connexion d'alice
aSocket = ss.accept();

//Tout le reste est pareil qu'avant
//idem
aIn  = aSocket.getInputStream();
aOut = aSocket.getOutputStream();

//envoyer puis recevoir un byte
int x=123; aOut.write(x);
int y = aIn.read();
```

Et les clés certificats?

- Ils doivent être créés avec keytool (format jks) ou openssl (pkcs12)

- Puis ils suffit de les référencer dans le code:
  - System.**setProperty**("javax.net.ssl.keyStore", "bob.jks");
  - System.**setProperty**("javax.net.ssl.keyStorePassword", "secret");
- Ou sur la ligne de commande:
  - java “-Djavax.net.ssl.trustStore=alice.jks”
    “-Djavax.net.ssl.trustStorePassword=changeit”
    your.main.Class

Autres couches

- Une fois que les octets entrent dans la socket Ils sont transportés vers l'autre socket.

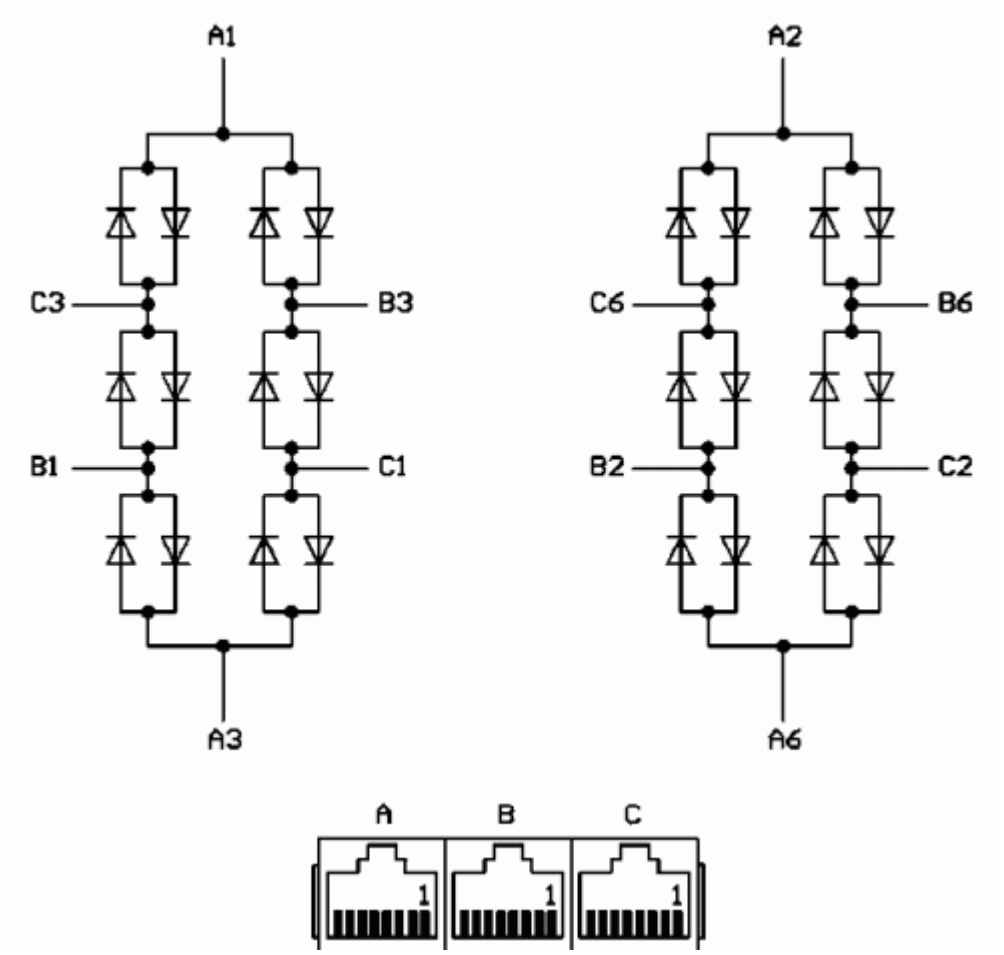
- Les octets peuvent transiter (encapsulés) dans les couches inférieures:
  - **IP:** On parle de routage
  - **Ethernet:** On parle de switching ou bridging
  - **Physique:**

Transférer des séquences physiques

- C'est le but des fils electriques ou des ondes
- Et ça marche tout seul!
  (hormis les erreurs, ce qui explique la présence des codes correcteurs)

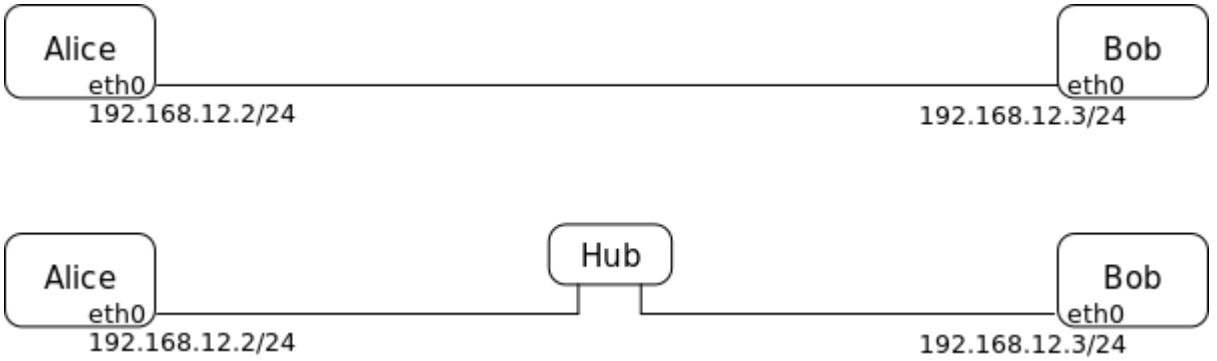
- Plus sérieusement, c'est le rôle des **hubs** et des **répétiteurs réseaux**

Hub réseau passif (et half duplex)



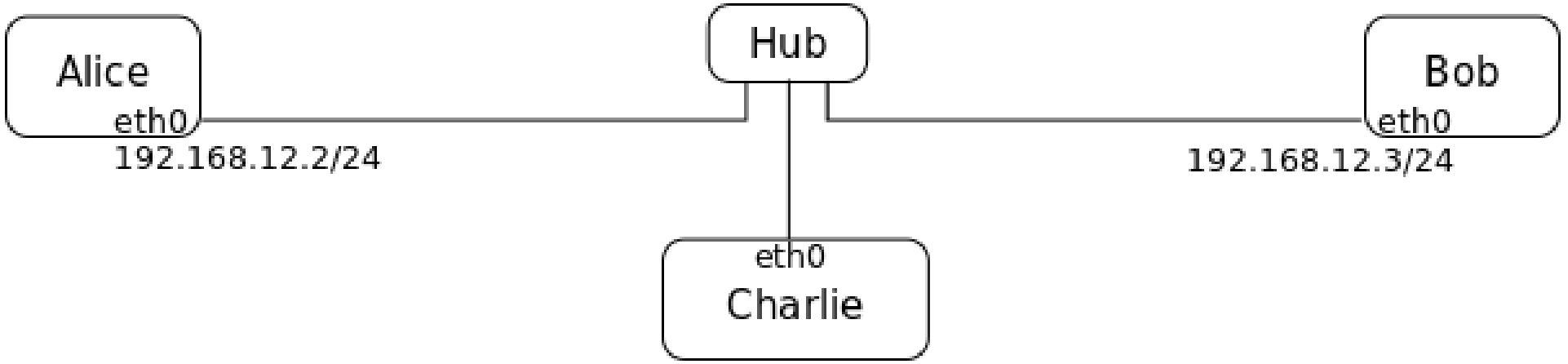
Il existe aussi des hubs actifs full duplex, très rares en filaires.
***Tous les réseaux wifi sont des hubs actifs** full duplex!*

Deux cas normaux d'utilisation



Question: Alice et Bob peuvent-il distinguer ces deux cas?

Un cas normal, certes…

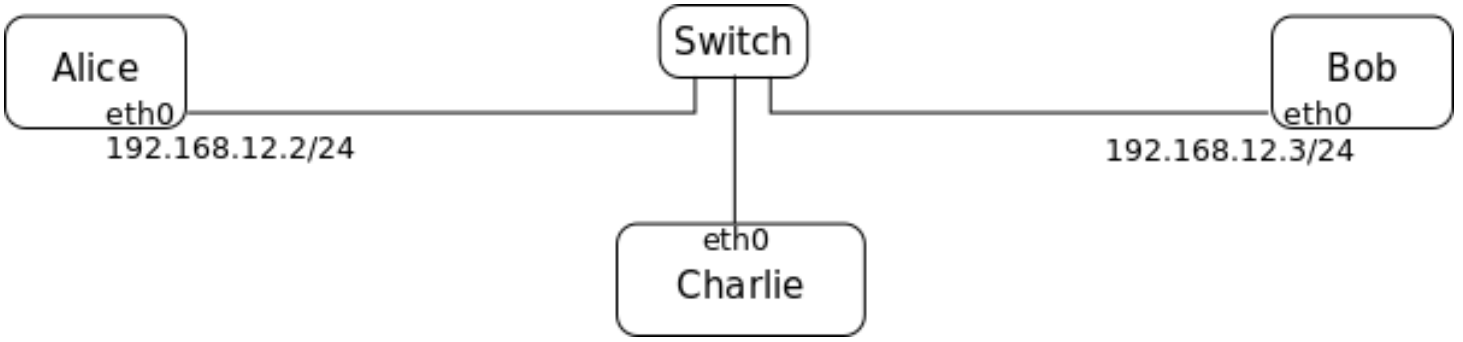


- Programmez cette attaque: alice et bob (et charlie) sur le même réseau wifi ad-hoc non chiffré.
- Est-ce détectable par Alice ou Bob?
- Proposez des contre-mesures?

Hubs

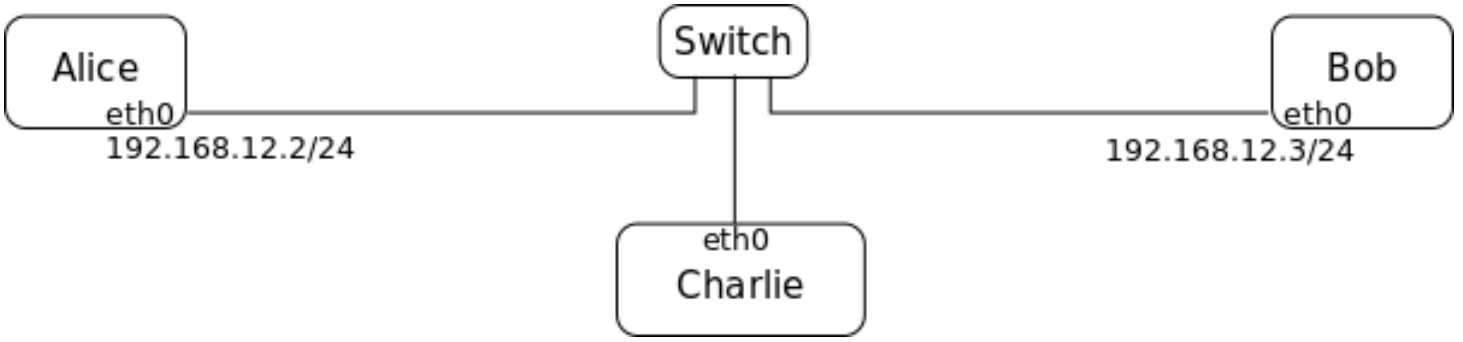
- **Inconvénient:**
  - Le réseau est vite saturé
  - Les attaques passives sont indetectables
  - La confidentialité doit être établie autrement (crypto, etc…)
- **Avantage:**
  - Toute attaque active est detectable!
  - Mais **detectable** ne veut pas dire **detecté!**

Switches



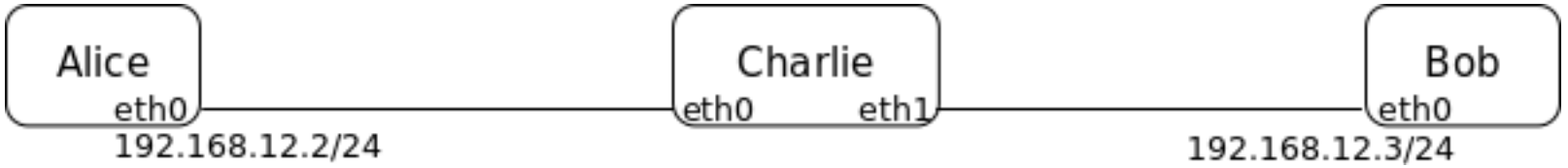
- Le switch opére sur la couche 2 (ethernet)
- Il connait les MAC de tout les appareils
- Il ne transfère une trame **qu'à son destinataire**
  - (sauf broadcast, multicast, ou exceptions)

Interception (presque) passive



- Programmez une attaque où Charlie intercepte et retransmet (sans les modifier) les données entre Alice et Bob.
  - Faites-le sans attribuer d'adresse IP à Charlie!
- Alice et Bob peuvent-ils detecter l'attaque?
- Proposer des contre-mesures.

Man (or monkey) in the middle

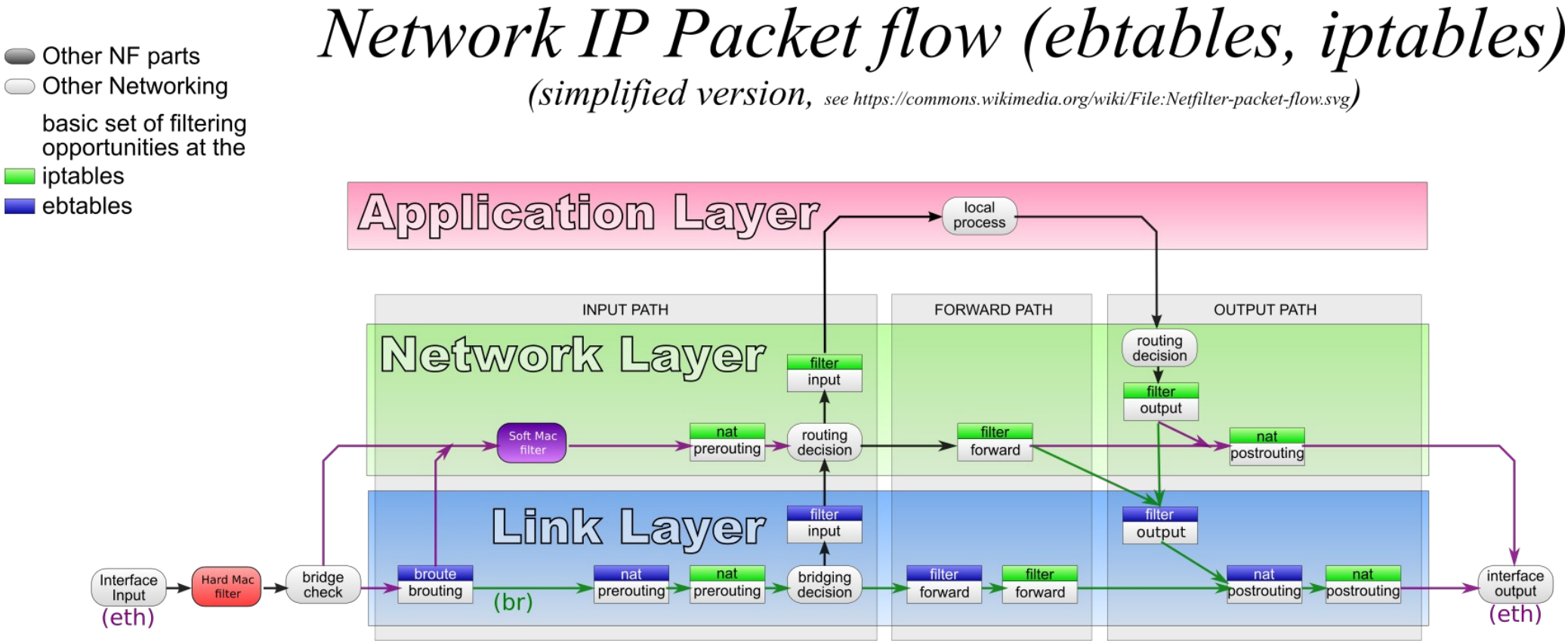


- Désormais, on considèrera ce scénario
  - Charlie est une machine linux avec deux interfaces
    - eth0 vers Alice
    - eth1 vers Bob.

Routeur passif niveau 3 (Proxy-Arp)

- Configurer Charlie pour forwarder les paquets IP entre eth0 et eth1.
- Pensez aussi à traiter les requêtes ARP entre alice et bob, qui passent maintenant via charlie!
- Cette attaque est-elle detectable? Proposer une contre mesure
  - Puis une contre-mesure contre la contre mesure ;)

Un peu de lecture



Mac filters

- Les trames ethernet qui arrivent sur une interface passent par deux mac-filters.
  - Ils rejettent toutes les trames (unicast) dont l'adresse mac destination n'est pas celle de l'interface.

    - Le hardware filter, sur la carte:
      il peut être désactivé (comment?)

    - Le software filter, en entrée de la couche IP:
      il ne peut **pas** être désactivé! (sans reprogrammer le noyau)

Conséquence des Mac filters

- A cause des mac filters, le seul moyen de faire passer les paquets est qu'Alice et Bob communiquent avec les adresses mac de Charlie.
  - (les mac-filters violent un peu le modèle OSI...)
- C'est la raison pour laquelle on parle de proxy-arp.
- L'attaque est donc detectable, sauf si Charlie copie les macs d'Alice et Bob.
  - Parfois, cela ne marche pas toujours…

Bridge passif niveau 2

- Configurer Charlie en un pure bridge.
- L'attaque est-elle detectable?

Garantir la confidentialité avec de la crypto

- Comme il existe des attaques passives indetectables, Alice et Bob vont utiliser du chiffrement pour communiquer:

  - Une variante utilisant Diffie Hellmann, puis AES

  - Une variante utilisant tout simplement TLS (SSL)

    (**Note:** l'implémentation java par défaut de SSL, contient une légère faiblesse que Charlie s'empressera d'exploiter!)

Proof of concept

- Et Charlie va prouver qu'il ne compte pas se laisser faire!
- Pour l'instant, Charlie configure son bridge sur le même sous-réseau qu'Alice et Bob.
- Programmez un man in the middle program pour les deux variantes (Diffie Hellmann et TLS)
  - Testez-les directement:
    Alice se connecte directement sur l'IP de Charlie:

    et Charlie prend temporairement
    la clé/certificat de Bob

Attaque Diffie-Hellmann indetectable

- En jouant sur iptables/ebtables,
- Rendre l'attaque Diffie-Hellmann indetectable:
  - Alice pense communiquer avec l'IP et la mac de Bob, et vice versa

Attaque SSL presque indetectable?

- Bob est un serveur public (en irelande), avec un certificat SSL valide émis par une vraie autorité racine.
  Ici: lab.algonics.net:4445
  - Vérifiez qu'alice l'accepte avec le truststore par défaut
- Charlie est la gateway du réseau d'Alice, et fournit entre autre DNS et DHCP.
- Charlie possède juste un certificat email smime valide, émis par une autorité racine du web.
- Montrer que dans ce cas, Charlie peut quand même faire un man in the middle où il retrouve la totalité des messages clairs, malgré la protection SSL!

Et contre-mesure!

- Quel est le problème que vous venez d'exploiter?
- Regardez par exemple:

  https://issues.apache.org/jira/browse/AMQ-5443

  - Pourquoi cette contre-mesure proposée empêche toute man in the middle attack?
  - À moins de corrompre à la fois le DNS ou la passerelle, **et** une autorité centrale de certification.

- Implémentez la contre-mesure, puis
  Vérifiez que l'attaque est bien detectée maintenant