

## X Window

### Sommaire

I.Introduction.....	1
II.Terminaux X.....	2
a)Terminaux X virtuels.....	2
b)La variable d'environnement DISPLAY.....	2
III.Qu'est qu'un serveur X et que fait-il ?.....	3
a)Lancement d'un serveur X.....	4
1Gestionnaire d'affichage (Display Manager).....	4
2Lancement manuel d'un serveur X.....	5
IV.Authentification des clients.....	7
a)Authentification par hôte.....	7
b)Authentification par cookie.....	7
1Fabrication du cookie.....	8
(1)Lancement manuel.....	8
(2)Avec gestionnaire d'affichage.....	9
2Transfert du cookie (ne devrait pas se faire sauf en environnement ultra protégé).....	10
3Utilisation du cookie.....	10
V.Transfert X11 par X11.....	10
VI.Transfert X11 par SSH.....	11
a)Utilisation sous Windows.....	11
b)Utilisation sous Linux.....	12
1Configuration du serveur.....	12
2Configuration du client.....	12
c)Fonctionnement (sous Linux).....	13
d)Mode Trusted et Untrusted.....	14
Bibliographie.....	15

### I. Introduction

L'affichage graphique sous Unix se fait par le biais du système X Window (ou plus simplement X). Ce système contrairement à celui de Windows se fait par le biais de connexions réseaux. Lors d'une utilisation locale, les communications se font par des sockets Unix, sortes de « fichiers tuyaux d'arrosage à double sens ». Un programme possède le premier bout et un autre le second. Tout ce qui est écrit d'un côté se retrouve de l'autre et réciproquement. C'est une abstraction permet l'affichage fenêtré sur différents systèmes d'exploitation et différents matériels.

Ce système X se compose :

- d'un serveur X qui gère l'affichage à l'écran des applications clientes et la transmission des entrées clavier et souris à ces mêmes programmes.
- de multiples clients qui sont les applications exécutées u des terminaux qui envoient des commandes d'affichage au serveur X et qui reçoivent les événements liés aux entrées.

En d'autre terme, cela signifie que l'interface Homme/Machine constituée de l'écran, du clavier et de la souris est bien séparé des programmes que l'utilisateur peut lancer et stockages de données. Ainsi, on peut avoir un terminal qui affiche des applications qui s'exécutent en fait sur un serveur se trouvant à l'autre bout du monde.

**On notera que les termes "serveur" et "client" sont utilisés dans le sens inverse dans lequel ils sont utilisés généralement avec d'autres applications client/serveur.**

## II. Terminaux X

Un terminal est simplement, un ordinateur qui va se connecter à un serveur pour demander une session et faire tourner ses programmes dessus. Il a juste à avoir un écran, une souris et un clavier, un moyen de faire tourner un serveur X (processeur et RAM) et une connexion réseau (principalement TCP/IP sur Ethernet) et c'est tout. C'est le serveur auquel il est connecté qui gère les programmes lancés et même le gestionnaire de fenêtre. Un terminal n'a pas besoin de lecteur de disquettes ou de disque dur ni tout autre moyen de stockage de données.

Il a deux solutions pour stocker son système d'exploitation :

- dans une ROM ce qui est très rare vu l'absence d'évolubilité
- en utilisant le mécanisme DHCP pour obtenir dynamiquement son adresse IP (en fonction de son adresse MAC) et le mécanisme de TFTP (ou NFS) pour récupérer l'image de son système d'exploitation depuis le serveur DHCP qui lui a donné son IP. TFTP est une version très allégée de FTP. En exemple de ce « boot réseau » est le système PXE d'Intel qui permet le téléchargement du système d'exploitation au démarrage.

Un terminal X n'a donc d'autre fonctionnalité que de faire de l'affichage et de la saisie clavier/souris et le tout en passant par le réseau TCP/IP sur Ethernet avec le protocole X.

### a) Terminaux X virtuels

Les terminaux X virtuels sont principalement le programme xterm. Ce sont de simple client X qui émulent une console texte en mode graphique.

### b) La variable d'environnement DISPLAY

Cette variable permet de dire vers quel serveur X se dirige les applications pour leur affichage et leur saisie. Elle peut avoir le format (les crochets indique un morceau facultatif) :

- `[nom_hôte]:affichage[.écran]` : `nom_hôte` est le nom de la machine sur laquelle tourne le serveur X qui affiche sur l'unité d'affichage numéro `affichage` dont fait parti l'écran numéro `écran`. Le serveur X écoute sur le port `6000+affichage` de la machine `nom_hôte`.
- `[nom_hôte/unix]:affichage[.écran]` : `nom_hôte` est le nom de la machine sur laquelle tourne le serveur X qui affiche sur l'unité d'affichage numéro `affichage` dont fait parti l'écran numéro `écran`. Le serveur X écoute sur le socket Unix `/tmp/.X11-unix/Xaffichage` de la machine `nom_hôte` qui est donc la seule à pouvoir y accéder.

Pour faire simple, une unité d'affichage c'est (à peu près) un utilisateur qui est connecté en session graphique (sur un environnement multi-utilisateur). Lorsque vous vous connectez, vous êtes sur **:0.0**. Si vous changez de console (Ctrl-F2), que vous vous loguez en mode texte et que vous lancez `startx -- :1`, vous lancez un second serveur X attaché à l'unité d'affichage **:1.0**. Il y a donc autant d'instance du serveur X et du gestionnaire de fenêtre que d'unité d'affichage.

**Dans les deux cas, si la partie entre crochets n'est pas présente, cela signifie que le serveur X tourne sur la machine locale.** Donc normalement votre variable DISPLAY (echo \$DISPLAY) devrait contenir « :0.0 »

C'est le contenu de cette variable qui indique aux programmes clients vers où il doivent se connecter pour affichage et saisie. On peut aussi passer cette information à un programme par l'option `-display` suivi d'un contenu dans le même format que DISPLAY.

**Note :**

Quand DISPLAY est à `:0.0`, il s'agit du socket Unix de l'affichage principal de la machine.

Mettre la variable DISPLAY à `localhost:n` ne signifie pas que l'on utilise un socket TCP/IP. Les applications traduisent cela en `localhost/unix:n`. Cela sera donc équivalent à un socket Unix.

### III. Qu'est qu'un serveur X et que fait-il ?



**Contrairement à ce que l'on pourrait penser, un serveur X n'est pas un gestionnaire de fenêtre. Ce n'est pas lui qui affiche le bureau Gnome mais le gestionnaire de fenêtre.**

Il permet de gérer l'affichage sur une unité ou écran, la saisie clavier et souris **mais pas le style ni le rendu (laissés au gestionnaire de fenêtre)**. Il assure aussi la transmission des saisies et la réception des commandes d'affichage par le biais du réseau.

**Et alors si ce n'est pas lui, qui est-ce qui fait mon bureau et mes fenêtres si jolies ?**

Eh bien ! Le gestionnaire de fenêtre !!!

Un gestionnaire de fenêtre est un client X comme un autre, il permet juste d'avoir un bureau, des fenêtres bien rangées...enfin bref, une interface conviviale à la Windows...

**Il assure aussi le rendu, la décoration, la sélection, le redimensionnement et le style de tout élément graphique.**

Par exemple, le gestionnaire de fenêtre GNOME est l'exécutable `gnome-session`.

## a) Lancement d'un serveur X

Il y a deux moyens d'obtenir un affichage graphique :

- les gestionnaires d'affichages (xdm, gdm ou kdm)
- le lancement manuel du serveur (x11, startx ou xstart)

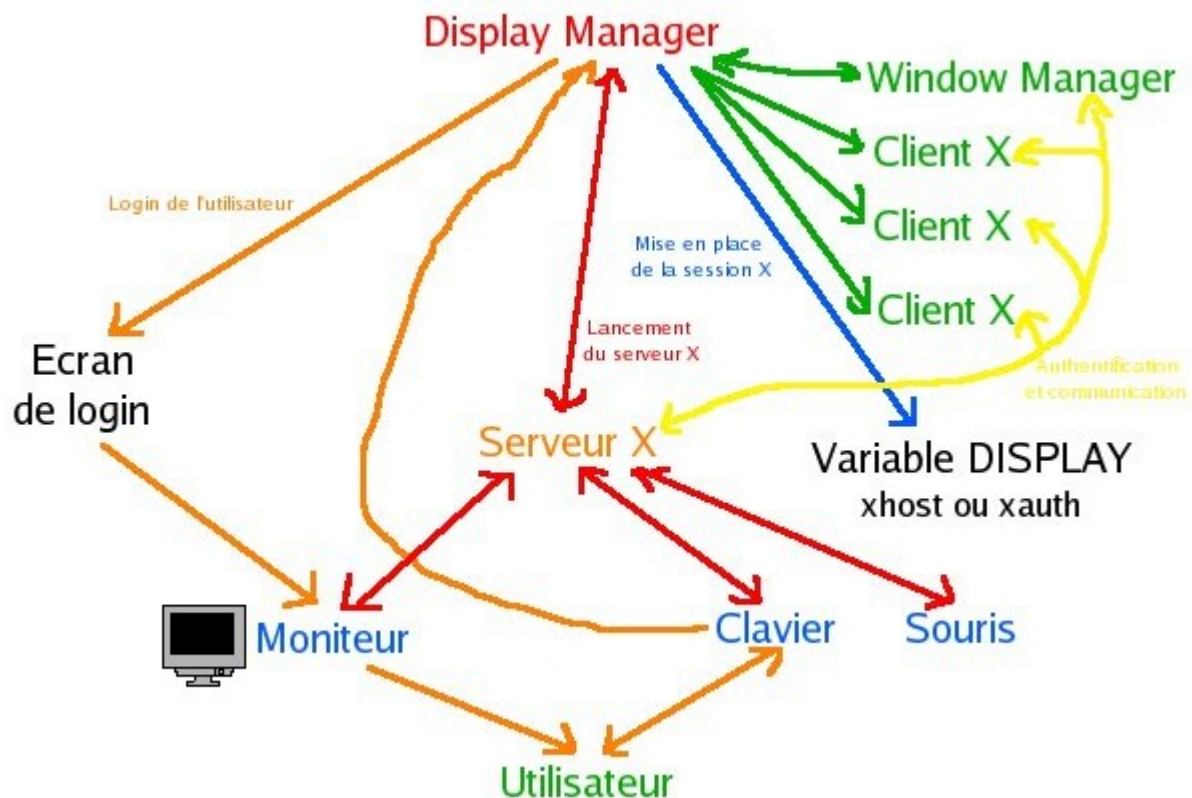
### 1 Gestionnaire d'affichage (Display Manager)

Il existe principalement trois gestionnaires d'affichage :

- xdm : c'est le serveur original maintenu par le consortium X
- gdm : c'est le serveur X de Gnome
- kdm : c'est le serveur X de KDE

Xdm, gdm ou kdm sont des gestionnaires d'affichage. Au premier abord, ils sont au mode graphique ce que « login » est au mode console : un moyen de se connecter à la machine et d'obtenir une session...graphique. Mais en fait, ils font beaucoup plus que cela.

Un gestionnaire d'affichage est un service Unix comme un autre, qui est démarré par le script rc au boot de la machine (ou par root lui-même) et qui tourne donc sous l'utilisateur root.



Une fois lancé, il est réaliser les actions suivantes :

- d'abord il doit présenter un écran graphique de login
- Si l'utilisateur est autorisé à se logger :
  - il va initialiser la sécurité des connexions clients par xhost ou xauth.

- il lance une instance du serveur X en le liant à l'unité d'affichage principale :0.0 (inscription dans la variable d'environnement DISPLAY). Il contrôle ce serveur X local.
- il va ensuite créer la session X :
  - régler l'environnement dont la variable DISPLAY fait partie
  - lancer en arrière plan les applications prévues dans la configuration (par exemple, \$HOME/.xsession)
  - enfin lancer au premier plan le gestionnaire de fenêtre (Window Manager).
  - Quand le gestionnaire de fenêtre s'arrête, il tue le serveur X
- Sinon, il affiche un message d'erreur et réaffiche l'écran de connexion.

Mais il peut aussi via le protocole XDMCP (X Display Manager Control Protocol) recevoir et contrôler des serveurs X locaux ou distants pour fournir un moyen d'authentification à la machine locale à ces serveurs demandeurs.

Il est donc capable de faire deux choses :

- gérer des serveurs X : il diffuse une invite de connexion à tous les serveurs X se trouvant dans sa liste `Xservers`.
- gérer des demandes d'écrans de login depuis d'autres serveurs X distants (dont la liste des autorisés est contenu dans le fichier `Xaccess`) :
  - demande ciblée : le serveur X distant demande directement l'écran de login
  - demande diffusées : le serveur X distant diffuse une demande et le premier serveur XDM qui répond gagne le droit d'envoyer son écran de login
  - demande d'une liste de serveurs XDM : le serveur X distant demande à un serveur XDM la liste de ceux qu'il connaît.

Un grand tutoriel est consacré à sa configuration sur

<http://www.ibiblio.org/pub/linux/docs/HOWTO/translations/fr/html-1page/XDM-Xterm.html>

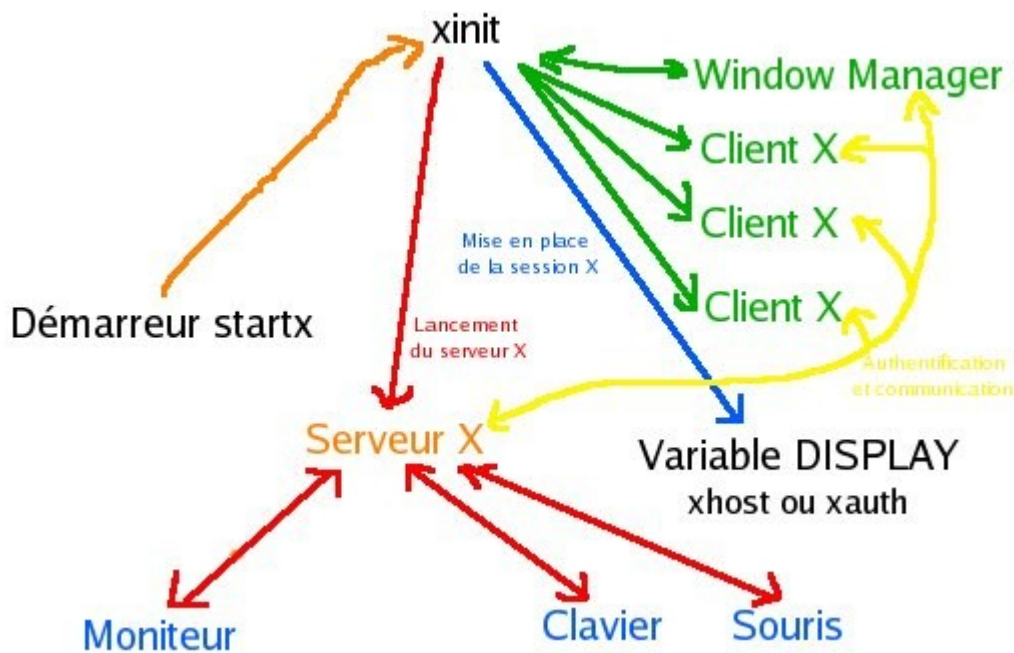
## 2 Lancement manuel d'un serveur X

On peut démarrer manuellement un serveur X pour plusieurs raisons :

- si on en peut pas le faire au démarrage (impossibilité ou mauvaise configuration)
- si on veut lancer plus d'un serveur X sur le même écran pour avoir plusieurs bureaux séparés (pour des utilisateurs différents par exemple)

Pour cela on utilise alternativement les commandes `x11` ou `startx` ou `xstart`. Ce sont tout trois des script qui appellent `xinit` pour lancer le serveur X et le client principal. Ils prennent, généralement, en argument :

- `-- :n` (où `n` est un nombre  $> 0$  indiquant un numéro d'unité d'affichage libre pour le serveur X lancé)



Ces scripts réalisent les opérations suivantes :

- il initialise les paramètres :
  - le client X et ses arguments par défaut passé en argument ou à défaut un `xterm`
  - le serveur X et ses arguments par défaut passé an argument ou à défaut `X :0.0`
- il initialise la sécurité `xauth` ou `xhost` en ajoutant les cookies ou nom d'hôtes nécessaires
- il lance la commande `xinit client_X args_client - serveur_X unite_affich args_serveur`
- `xinit` lance le fichier `$HOME/.xserverrc` ou le serveur X sur l'unité d'affichage passée et avec les arguments passés
- `xinit` lance le fichier `$HOME/.xinitrc` ou le client X passé en argument ou le fichier `/etc/X11/xinit/xinitrc`
  - dans le premier et troisième cas, le fichier `.xinitrc` doit lancer une série de programme en arrière plan pour ne pas empêcher les suivants de se lancer et le dernier (souvent le gestionnaire de fenêtre) au premier plan pour empêcher la session X de se fermer
  - dans le second cas, le client X est lancé au premier plan
  - Dans tous les cas, une fois l'application de premier plan terminée, `xinit` se termine
- il attend la fin de `xinit`
- il supprime la sécurité qu'il a ajouté avant le lancement

On aura donc souvent l'arborescence suivante :

`startx --> xinit --> gnome/kde`

## IV. Authentification des clients

Il existe deux types d'authentification pour les serveurs X :

- l'authentification par hôte : **sécurité quasi nulle**
- l'authentification par cookie (donc par utilisateur)

### a) Authentification par hôte

Xhost permet une authentification basées sur les noms d'hôtes. Le serveur X maintient une liste des hôtes qui sont autorisés à se connecter à lui. **On peut aussi complètement désactiver ce contrôle d'accès ce qui revient à AUTORISER TOUT INTERNET à voir ce que l'on voit et savoir ce que l'on tape au clavier et comment on déplace la souris** si la machine sur laquelle tourne le serveur X est exposée sur Internet.

On peut voir et modifier la liste des hôtes autorisés du serveur X avec la commande `xhost`.

Pour ajouter un hôte autorisé, on tapera

```
utilisateur_connectés$ xhost +nom_hôte
```

Ceci permet d'autoriser toutes les connexions à partir de l'hôte `nom_hôte`. Une fois que le client X a réalisé sa connexion et affiche une fenêtre, par sécurité, on devrait supprimer les l'autorisation pour d'autres connexions par la commande :

```
utilisateur_conTransfert X11 par nectés$ xhost -nom_hôte
```

**À NE PAS FAIRE à moins d'être totalement sûr de son voisinage réseau** : désactiver la vérification des hôtes autorisés :

```
utilisateur_connectés$ xhost +
```

Ceci désactive la vérification des accès autorisés des hôtes et donc permet à *tout le monde* de se connecter. Vous ne devriez *jamais* faire cela sur un réseau où vous n'avez pas confiance dans *tous* les utilisateurs On peut réactiver la vérification des hôtes avec la commande :

```
utilisateur_connectés$ xhost -
```

Note : `xhost -` *ne supprime pas* tous les hôtes de la liste d'accès car vous ne pourriez plus vous connecter de n'importe où, pas même de votre hôte local et donc vous perdriez votre affichage.

**ATTENTION : Xhost est vraiment un mécanisme peut sûr pour deux raisons :**

- il ne distingue pas les utilisateurs d'un même hôte mais simplement la machine elle-même
- les nom d'hôtes (et donc les IP) peuvent être très facilement falsifiée par des techniques de piratage comme l'IP spoofing et provoquer des DoS.

### b) Authentification par cookie

Xauth autorise l'accès à tous ceux qui connaissent le bon « secret ». On appelle un tel « secret » un enregistrement d'autorisation ou plus simplement un cookie. Ce mécanisme d'autorisation s'appelle

en réalité le MIT-MAGIC-COOKIE-1.

Tous les cookies d'un utilisateur (pour toutes les unités d'affichage auquel il a accès) sont stockés dans le fichier `.Xauthority` de son répertoire de connexion. Ce fichier ne doit être accessible que par l'utilisateur dont c'est le répertoire de connexion, autrement dit `rw----- (600)`.

La commande chargée de gérer ce type d'authentification et donc les fichiers `.Xauthority` est la commande `xauth`.

La variable d'environnement `XAUTHORITY` permet de changer le nom et le chemin de ce fichier de cookie mais c'est rare. Pour connaître le nom de ce fichier, il suffit de taper `xauth -v`.

La séquence d'authentification est la suivante :

- Au démarrage, le serveur X lit son cookie dans le fichier qui lui est indiqué par l'argument `-auth`.
- Ensuite, le serveur ne permet la connexion que des clients qui connaissent le même cookie. Quand on change un de ses cookies dans le fichier `.Xauthority`, le serveur n'a pas de moyen d'en connaître la modification. On peut se retrouver dans ce cas, incapable de se connecter à nouveau. Pour mettre à jour la liste des cookie dans le serveur, il faut exécuter la commande `xhost add`.

Les serveurs les plus récents peuvent générer des cookies à la volée pour des clients qui le demandent. Cependant de tels cookies ne sont stockés que dans le serveur et pas dans le fichier `.Xauthority` du client.

Xauth permet donc de palier aux lacunes de Xhost :

- il permet d'identifier des utilisateurs particuliers sur des machines particulières
- il n'est pas sensible à l'usurpation d'adresse IP

## 1 Fabrication du cookie

Les cookies sont des morceaux de texte (hash) de 128bits.

Si vous voulez utiliser `xauth`, vous devez lancer le serveur X avec l'argument `-auth fichier_de_cookie`.

### (1) Lancement manuel

Si l'on utilise `startx` ou `xstart` ou `x11`, on doit ajouter les commandes permettant l'ajout de cookie :

Par exemple dans `/usr/X11R6/bin/startx` :

```
mcookie|sed -e 's/^/add :0 . /'|xauth -q
xinit -- -auth "$HOME/.Xauthority"
```

Ou encore :

```
xauth -q << EOF
```

```
add unite_affich . `mcookie`  
EOF
```

mcookie est un programme permettant de générer un cookie xauth à partir de données aléatoire comme son PID, son PPID, /dev/urandom, /proc, le timestamp courant...

Sinon, on peut écrire un script ~/.xserverrc. Il sera alors exécuté par xinit au lieu du véritable serveur X. Il faudra lancer le serveur X véritable à partir de ce script avec les arguments adaptés.

Voici un exmple :

```
#!/bin/sh  
mcookie|sed -e 's/^/add :0 . /'|xauth -q  
exec /usr/X11R6/bin/X :0 -auth "$HOME/.Xauthority"
```



## (2) Avec gestionnaire d'affichage

Si l'on utilise xdm, gdm ou kdm pour gérer ses session X, il mettra tout seul le cookie dans \$HOME/.Xauthority et en argument du serveur X qu'il lance (-auth).



## 2 Transfert du cookie (ne devrait pas se faire sauf en environnement ultra protégé)

On pourrait transférer le cookie généré dans un des vos répertoires distants afin de pouvoir exécuter des programmes à distance et voir le résultat sur votre écran. Pour cela, il faut générer un cookie de la façon suivante :

```
#!/bin/sh
mcookie|sed -e 's/^/add :0 . /' -e p -e "s/:/$HOST&/"|xauth -q
exec /usr/X11R6/bin/X "$@" -auth "$HOME/.Xauthority"
```

Cela permet de générer un cookie pour l'affichage local :0.0 et pour l'affichage depuis un serveur extérieur sur l'hôte local (dont \$HOST contient le nom DNS).

**Cependant cette méthode n'est pas sûre car les données transitent en clair sur le réseau. Il est préférable de faire de la redirection X11 par SSH.**

## 3 Utilisation du cookie

Une application X, telle que gedit, pour pouvoir interagir avec l'utilisateur devra aller automatiquement voir le cookie dans ~/.Xauthority pour s'authentifier auprès du serveur X.

Si malgré tous les risques, vous souhaitez des connexions X en clair entre deux machines d'un réseau, il faut faire attention de bien mettre le nom complet de la machine serveur X afin que le cookie soit valable pour celle-ci et pas pour seulement localhost. De plus, cela pourrait provoquer des erreurs car localhost fait toujours référence à la machine SUR LAQUELLE s'exécute le code.



## V. Transfert X11 par X11 sous Linux

On cherche ici à exécuter des programmes sur un serveur distant et voir le résultat sur notre écran. Il est à noter que dans ce cas, notre machine locale se résume au rôle d'un simple terminal.

**Attention : tout le trafic est en clair ce qui signifie que tout le monde peut voir ce que vous voyez et lire ce que vous tapez.**

Pour ce faire, il suffit d'exporter la variable DISPLAY sur la machine distante :

```
[utilisateur]$ export DISPLAY=IP_ou_nom_DNS_machine_local:n
```

Et de lancer un serveur X sur la machine locale qui écoute sur le port 6000+n :

```
[utilisateur]$ startx -- :n
```

où  $n$  est un nombre supérieur à 1 si l'on veut avoir l'affichage distant sur un autre serveur X local ou 0 si on veut mélanger les applications X locales et distantes.

Il faut aussi donc inclure une règle iptables pour autoriser ce trafic :

```
[root]# iptables -A INPUT -p tcp --dport 6000+n -s
IP_machine_distante -j ACCEPT
[root]# iptables -A OUTPUT -p tcp --sport 6000+n -d
IP_machine_distante -j ACCEPT
```

## VI. Transfert X11 par XDMCP

### a) Côté serveur

Dans le fichier `/etc/X11/gdm/gdm.conf`, dans la section `[xdmcp]`, mettre

```
Enable=true
```

```
Port=177
```

Eventuellement dans un fichier `/etc/X11/gdm/Xaccess`, mettre sur une ligne seule :

```
*
```

En ce qui concerne iptables, il faut savoir que X11 par XDMCP utilise

- le port UDP 177 : pour XDMCP
- le port TCP 6000 : pour les communications X

Il en résulte les règles suivantes :

```
iptables -A INPUT -p udp --dport 177 -j ACCEPT
iptables -A OUTPUT -p udp --sport 177 -j ACCEPT
iptables -A INPUT -p tcp --dport 6000 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 6000 -j ACCEPT
```

**Note très importante :** tout le trafic se fait en clair. Il faut donc restreindre XDMCP à une liste d'hôtes connus et autorisé à se connecter à distance par X11. Il est aussi impératif que le réseau local sur lequel se fait XDMCP soit dûment firewallé. C'est pour cela que XDMCP est désactivé par défaut. **Si l'on est dans un environnement peu sûr, on préférera le transfert X11 par SSH.**

### b) Utilisation sous Windows avec Xming et GDM

**Note :** Si vous utilisez « Microsoft Services for Unix » supprimer la variable d'environnement 'XKEYSYMDB' avant de lancer Xming sinon le clavier ne marchera pas (Panneau de configuration/Système/Avancé/Variables d'environnement).

Lancer XLaunch :

- sélectionner le type d'affichage que vous souhaitez : fenêtres multiples, une fenêtre contenant toutes les autres... laisser 0 dans « Display number » puis cliquer sur « Suivant »

- sélectionner «Open session via XDMCP » puis Suivant
- si vous connaissez l'hôte XDMCP à utiliser :
  - sélectionner et mettre son nom DNS ou son IP dans « Connect to host »
  - si l'hôte XDMCP relaie les connexions vers un autre hôte, cocher indirect
  - si vous ne connaissez pas votre hôte XDMCP, sélectionner « Search for hosts (broadcast) »
- cliquer sur Suivant puis Suivant
- cliquer sur « Save configuration » pour sauvegarder la configuration de Xming.
- Cliquer sur « Terminer » pour lancer le serveur X

## c) Utilisation sous Linux

# VII. Transfert X11 par SSH

## a) Utilisation sous Windows

Tout d'abord, contrairement à ce que l'on peut obtenir par la connexion directe en clair entre le serveur X et le client X, avec cette redirection, on ne voit que les applications que l'on lance depuis la ligne de commande et pas tout le bureau.

- Premièrement, il faut que le paquetage SSH soit installé. Dans Linux, il s'agit des paquetages OpenSSH et OpenSSL.
- Deuxièmement, il vous faut un client SSH Windows comme PuTTY. C'est un client SSH libre.
- Troisièmement, il vous faut un serveur X sous Windows. Il existe par exemple Xming ou Exceed.

Réalisation de la redirection X11 par SSH :

- Ouvrez le programme **putty.exe** en double-cliquant dessus. Cela va lancer l'interface.
  - Premièrement, renseignez les informations de connexion dans le champ “Host Name (or IP address)” avec le nom de l'hôte distant ou son adresse IP et sélectionnez SSH (SSH utilise le port 22).
  - Dans l'arbre “Category”, trouvez la branche “Connection”.
    - Développez SSH et vous verrez l'item “X11”.
    - Cliquez sur “Enable X11 forwarding” (autoriser le transfert de port). Il configurera l'affichage graphique par défaut à localhost:0.
  - À présent, retournez à la fenêtre “Session” et sauvegardez cette session sous le nom que vous voulez.
- Si Exceed est votre serveur X local sous Windows, pour les autres serveurs X, la configuration est analogue.
  - Ouvrez Xconfig de votre répertoire Exceed.

- Dans votre écran Screen Definition (définition de l'écran), passez le "Window mode" à "Multiple" et sauvegardez-le.
- Appelez ensuite votre icône "Communication" et définissez le mode "Startup" à "Passive".
- Si vous utilisez Xming, lancer Xlaunch puis :
  - sélectionner le type d'affichage que vous souhaitez : fenêtres multiples, une fenêtre contenant toutes les autres... laisser 0 dans « Display number » puis cliquer sur « Suivant »
  - sélectionner « Start no client » puis Suivant
  - cliquer sur Suivant
  - cliquer sur « Save configuration » pour sauvegarder la configuration de Xming.
  - Cliquer sur « Terminer » pour lancer le serveur X
- À la première connexion, PuTTY vous demandera si vous désirez conserver la clé de sécurité. ("Yes" (oui) est le choix par défaut).
- Une fois connecté, lancez Exceed. Il demeurera en tâche de fond.
- Vous pouvez à présent exécuter n'importe quelle application X; l'application X devrait être transmise à travers SSH à votre écran local. Par exemple :

```
$ gedit &
```

Vous devriez voir l'éditeur Gedit s'afficher sur votre écran local.

**Note** : si vous utilisez Cygwin, vous n'avez pas besoin de PuTTY. Il suffit que vous installiez le paquetage X11 et openssh (dans Net). Ensuite, lorsque vous êtes dans le terminal Cygwin (noir par défaut comme un console) taper startx puis dans la nouvelle console, vous pouvez suivre la procédure Linux ci-dessous.

## b) Utilisation sous Linux

Il est recommandé d'utiliser l'option -C pour permettre la compression des données circulant dans le canal.

On peut l'utiliser de plusieurs manières :

- en ligne de commande unique : `ssh -f -X -C serveur application_graphique`. Cela permet de lancer en tâche de fond (-f) l'*application\_graphique*. L'affichage se fera sur votre écran. La compression (-C) permettra d'accélérer l'affichage. La connexion SSH sera automatiquement fermée à la fermeture de l'*application\_graphique*.
- On lance un ssh normal : `ssh -X -C serveur`. Puis une fois authentifié, on lance les programmes en tâches de fond : `[user distant]$ application_graphique &`.

Il est nécessaire de l'assurer (en cas de problème) que les précautions suivantes sont prises :

- S'assurer que la variable **DISPLAY** est bien initialisée sur la machine locale (dont vous avez la console sous les yeux), par exemple à **:0.0**
- S'assurer que le serveur SSH auquel on se connecte accepte de transférer les connexions X-Window. Voir section configuration.
- S'assurer que sur le client SSH le transfert X11 est activé. Voir configuration ou option -X.

- Sur la machine distante il faut bien faire attention de NE PAS y modifier la variable d'environnement DISPLAY initialisée par SSH
- SSH protège l'accès à ce pseudo-DISPLAY en plaçant une entrée adéquate dans votre fichier **\$HOME/.Xauthority** sur la machine distante; il ne faut donc pas effacer cette ligne aussi longtemps que le DISPLAY correspondant est actif.

## 1 Configuration du serveur

Pour autoriser les clients SSH à rediriger les programmes graphiques sur leurs écrans (redirection X11), les directives suivantes doivent prendre la valeur *yes* :

- X11Forwarding doit être à *yes* pour autoriser la redirection X11 par le canal SSH
- X11UseLocalhost doit être à *yes* afin d'éviter les tentatives de hacks sur les clients liés à l'existence d'un serveur X proxy en sortie côté client du canal SSH. Cela n'autorise le serveur X proxy qu'à attendre les connexions sur la boucle locale (inaccessible depuis l'extérieur), ce qui permet au canal de communiquer avec lui mais pas au voisinage du client. Ceci peut se rapprocher en opposé à l'option *-g* qui permet d'exposer à l'extérieur les redirections de ports.

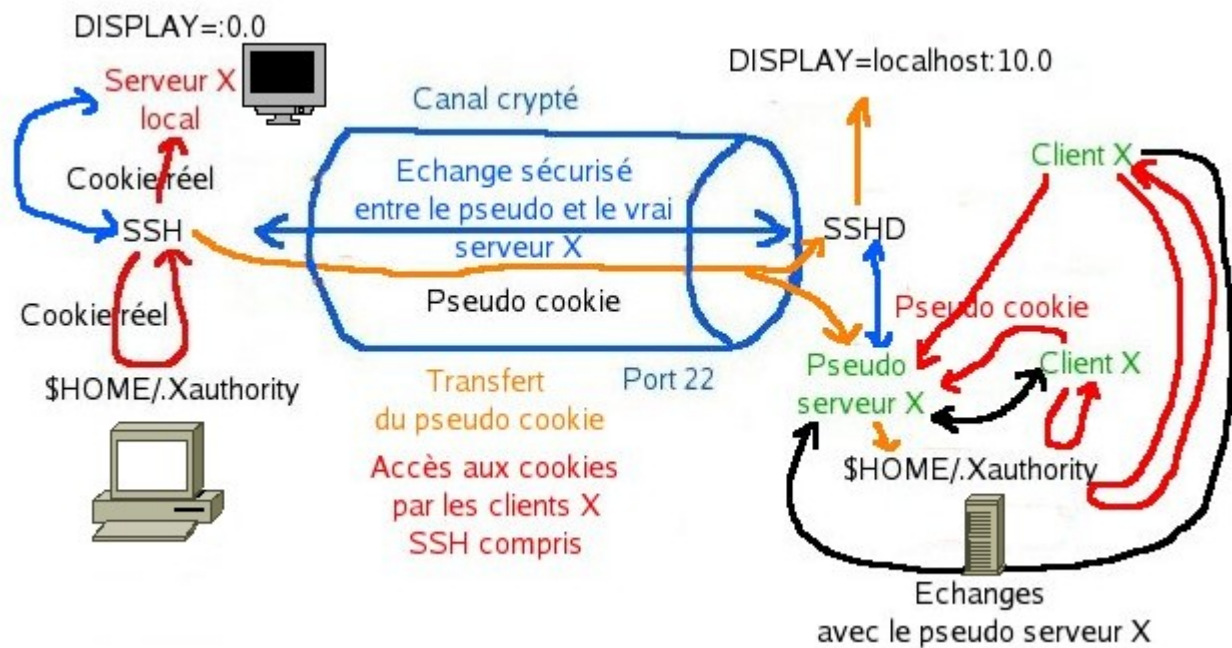
## 2 Configuration du client

Pour autoriser les clients à recevoir la redirection X11, le client doit :

- soit configurer la directive `ForwardX11 yes` dans le fichier `/etc/ssh_config` ou autre fichier de configuration personnel
- soit passer l'option `-X` en ligne de commande de `ssh`

## c) Fonctionnement (sous Linux)

**D'abord, petite précision, pour X Window, le concept de client et serveur est inversé. Le client X est tout programme graphique lancé par le client sur le serveur SSH. Et le serveur X est le serveur tournant que le client qui permet l'affichage local au client. Voir tutoriel sur X.**



Si le terminal depuis lequel on a appelé ssh dispose d'un affichage graphique :

- **il extrait le cookie** du fichier `$HOME/.Xauthority` afin de pouvoir s'authentifier plus tard pour communiquer avec le serveur X local
- **il génère un cookie aléatoire** pour fournir aux clients X sur le serveur SSH distant. Ce cookie servira à l'authentification des clients X sur le pseudo serveur X lancé sur le serveur SSH. En effet, il est plus sûr de ne pas transmettre le cookie réel d'authentification local pour ne pas permettre à un éventuel intrus de récupérer ce cookie (par le système de fichier distant si les permissions sont mal réglées) et de se connecter directement au serveur X du client.
- **il transporte ce cookie** au travers du canal sécurisé et le transmet au serveur SSHD
- **celui-ci lance un pseudo serveur X** auquel les programmes graphiques lancés sur le serveur SSHD vont se connecter
- **il place le cookie aléatoire reçu dans le fichier .Xauthority dossier de l'utilisateur connecté**
- **il règle la variable d'environnement DISPLAY** pour pointer vers le pseudo serveur X lancé pour l'occasion (par exemple localhost:10.0 si le pseudo serveur X écoute sur le port 6010 du serveur SSH)
- ainsi, quand un programme veut se connecter pour recevoir les entrées clavier/souris et envoyer des commandes d'affichage au client :
  - **il récupère le cookie aléatoire** dans le fichier `.Xauthority`
  - **il l'envoie au pseudo serveur**
  - **celui-ci le fait transiter par le canal sécurisé**
  - le client SSH vérifie que c'est bien le cookie aléatoire qu'il a généré
    - si oui, il autorise le client X à se connecter au serveur X du client pour affichage et entrées
    - sinon, il s'agit sûrement d'un intrus qui veut se connecter au pseudo serveur pour accéder au client

- une fois l'authentification réalisée, **il retransmet tout le trafic** venant
  - du client X au serveur X : le client X envoie ses données au pseudo serveur X de SSHD qui les envoie par le canal sécurisé. SSH les reçoit et les retransmet au vrai serveur X du client
  - du serveur X vers le client X : le serveur envoie ses données à SSH qui les transmet au pseudo serveur X de SSHD qui lui-même les retransmet au client X.

**Note sur la sécurité de l'opération : il est très important que l'utilisateur connecté soit le seul (à part root) à pouvoir lire le fichier .Xauthority situé dans son répertoire de connexion, autrement dit avec les droits rw----- (600).** Sans cela, n'importe qui qui aurait accès à ce fichier pourrait se connecter au pseudo serveur X du serveur SSHD et ainsi voir ce que voit le client SSH saisiés souris et clavier compris.

Note technique :

- le pseudo serveur X est partie intégrante du serveur SSHD.
- du côté du client SSH, la connexion au serveur X local se fait par le biais d'un socket Unix (par exemple le fichier /tmp/.X11-unix/X0).
- du côté du pseudo serveur X et des clients X lancés sur le serveur SSHD, les connexions se font en TCP/IP sur le port *6000+unité\_affichage\_choisie* sur l'interface lo (127.0.0.1). Ceci est dû au fait que seul root peut créer ce type de socket dans ce répertoire.

## d) Mode Trusted et Untrusted

Le modèle de sécurité X11 fait une distinction entre des clients « trusted » et « untrusted ». Sans rentrer dans le détail, les clients X11 « untrusted » ne peuvent pas agir avec les fenêtres et ressources possédées par des clients « trusted ».

Depuis la version 3.8, OpenSSH supporte la sécurité X11 et fait une redirection X11 « untrusted » par défaut. Cependant, la plus part des applications quelles sont des clients X « trusted » et peuvent ne pas fonctionner correctement ou crasher de manière inattendue en mode « untrusted ». Il est seulement nécessaire de configurer la redirection X11 en mode « trusted » dans les cas précédents. Cela peut se faire de deux façons différentes suivant le caractère permanent à donner ou pas :

- Mode « trusted » au cas par cas : utiliser le commutateur `-Y` à la place de `-X`
- Mode « trusted » pour toutes les connexions : ajoute la ligne suivante au fichier `ssh_config` :  
`ForwardX11Trusted yes`

## Bibliographie

[X.Org Foundation](#)

[XFree86® Home to the X Window System](#)

[The X Window User HOWTO](#)

**XDM et les terminaux X, mini-HOWTO**

mini-HOWTO : Comment exécuter des applications X à distance

**GDM - GNOME Display Manager**

**Le manuel de Kdm - KDE Display Manager: Installation de Kdm**

<http://www.straightrunning.com/XmingNotes/>