CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

# *Cours d'administration réseau TCP-IP et sécurité : Quelques applications fondamentales*

Denis Pugnère

CNRS / IN2P3 / IPNL

d.pugnere @ ipnl.in2p3.fr

# ICANN

ICANN : Internet Corporation for Assigned Names and Numbers

gère les adresses IP, l ’organisation de protocoles, les noms de domaines, et les serveurs « root » au niveau mondial.

La majorité de ces tâches étaient sous la responsabilité du gouvernement des U.S.A

3 Supporting Organizations (S0) : Recommendations sur la politique d ’Internet et sa structure :

Address Supporting Organization (ASO) : concerné par l ’adressage IP

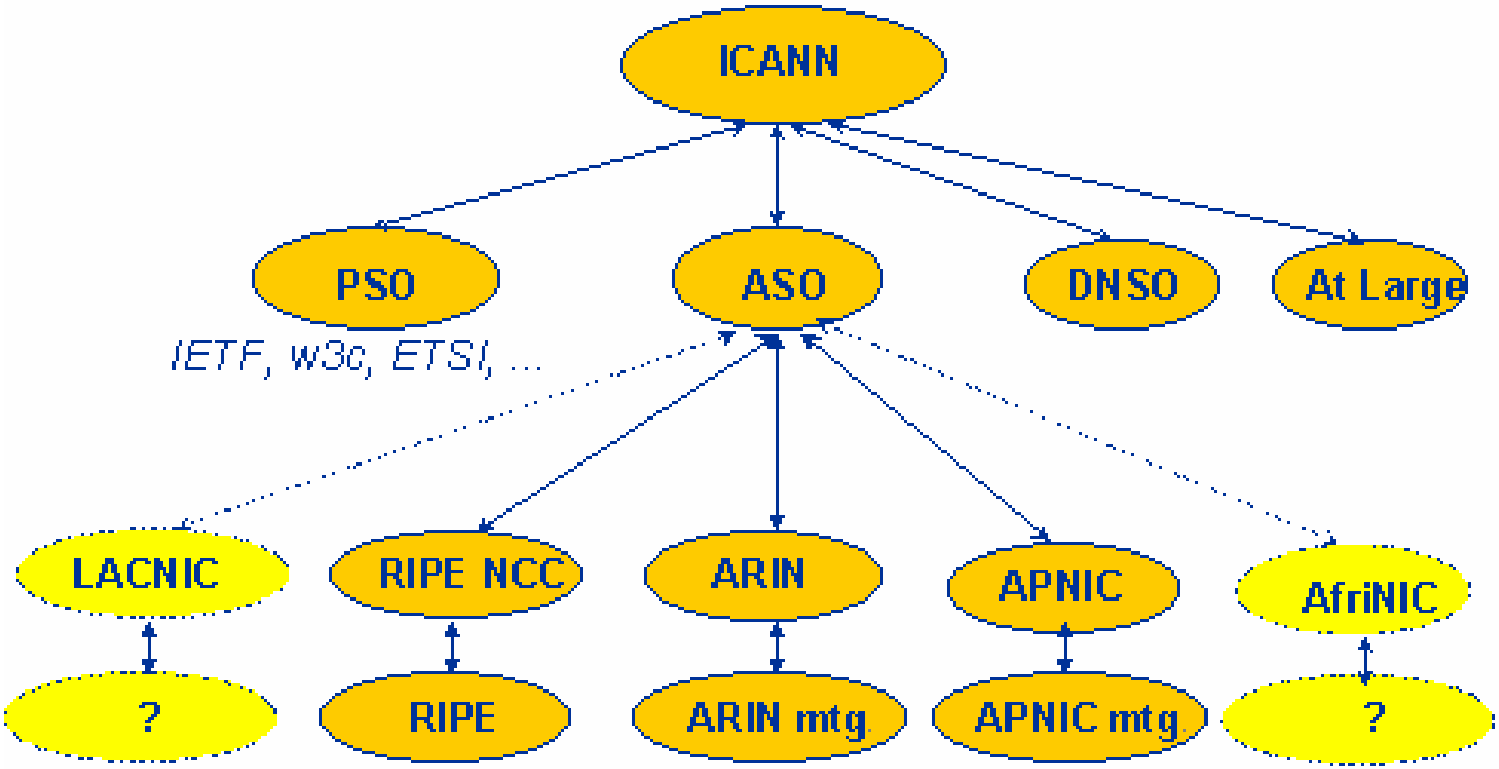
Domain Name Supporting Organization : (DNSO) : concerné par la gestion du système de noms (DNS) : gTLD, ccTLD, registrars

Protocol Supporting Organization (PSO) : concerné par l ’affectation des paramètres des protocoles Internet

Le GIP Renater est actif dans ICANN.

Le Nic Français (afnic) est actif dans le DNSO de l ’ICANN

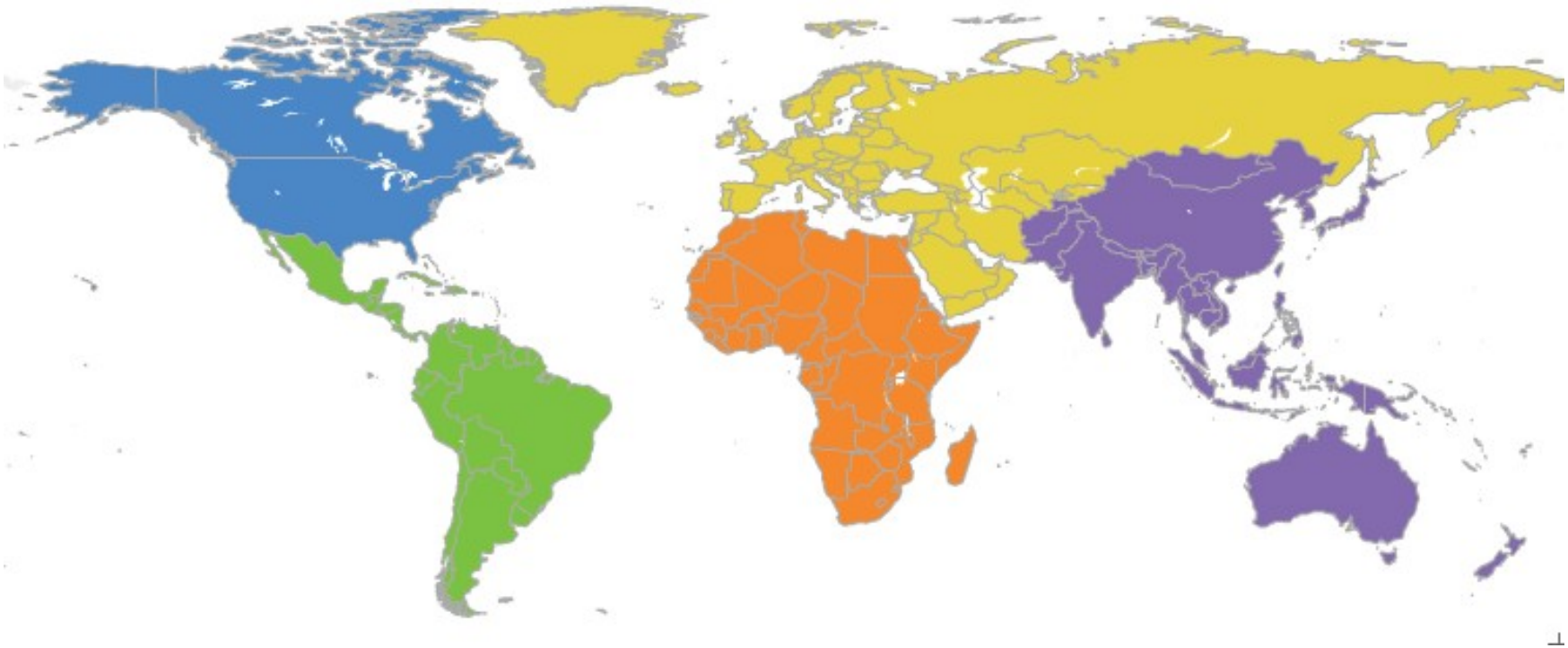
# Organisation de l'ICANN



## Regional Internet Registries

**Hover for more information. Drag or click to zoom. Boundaries shown are not necessarily authoritative.**

ARIN LACNIC AfriNIC RIPE NCC APNIC



# Les Regional Internet Registeries

RIPE NCC (Réseaux IP Européens Network Coordination Centre) : Europe Est/ Ouest, Afrique du Nord

ARIN (American Registry for Internet Numbers) : Amérique Nord/Sud, Afrique du Sud

APNIC (Asia Pacific Network Information Centre) : Asie

LACNIC (Latin America and Caribbean Network Information Centre) : Amérique du Sud et Amérique Centrale

AfrNIC (Africa Network Information Centre) : Afrique

Le RIPE est l'organisme qui gère les adresses IP au plan européen. Il est un des 5 RIR, son activité (au plan européen) :

allocation adresses IP

allocation « interdomain routing identifiers » (n° AS BGP)

allocation reverse DNS (in-addr.arpa et ip6.int)

gestion base Whois RIPE

Le GIP Renater participe aux travaux de RIPE.

# DNS

Domain name system : RFC 1032, 1033, 1034 et 1035

Système mondial, coopératif, cohérent et hiérarchisé de nommage.

Gestion décentralisée des informations de la base de données globale :

Usage général indépendant des types d'applications et du type de machines : du micro au mainframe !

Les équipements communiquent grâce à leur adresse IP.

Seules les applications utilisent les noms des équipements :

A une adresse IP peut correspondre un ou plusieurs noms (alias)

Un nom doit être unique au monde. Exemple :
@Ip : 195.83.84.20 = pegase.Igh.Cnrs.Fr (alias mail.Igh.Cnrs.Fr)

case insensitive

Deux mécanismes de résolution de noms :

Résolution normale : nom -> adresse(s) ?

Résolution inverse : adresse -> nom(s) ?

# DNS

Les TLD (top level domains : exemple .Com, .Mil, .Net, .Edu, .Uk, .Fr) délèguent la résolution des SLD (sous domaines) à d 'autres (exemple : cnrs.Fr)…

ccTLD (country code) :composé de 2 lettres : ISO 3166 : fr, us, uk… (+ de 240)

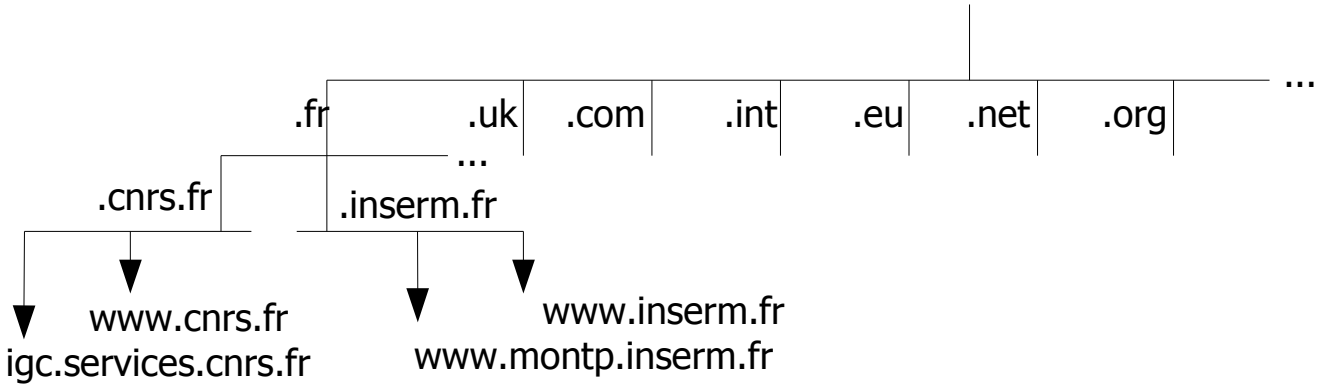
RTLD (Regional) : .int, .eu, .asia

gTLD (generic) :

en 1980 : .int, .edu, .mil, .gov, .com, .org, .net

En 2000 : unsponsored (.biz, .info, .name, and .pro), sponsored (.aero, .coop, and .museum)

Spécial (raisons techniques) : in-addr.arpa et ip6.int



# DNS : Les serveurs

Les serveurs primaires détiennent l'origine des informations d'une zone (ensemble de machines) grâce à des fichiers de configuration

/etc/named.conf

/etc/named.data/db.hosts

/etc/named.data/named.local

Les serveurs secondaires (esclaves) obtiennent leurs informations (périodiquement ou à la demande) par un échange (tranfer de zone) avec le serveur primaire

Les clients : demandent une résolution de nom (via un "resolver" (gethostbyname, gethostbyaddress))

# DNS : les enregistrements

| Type | Signification | Valeur |
|---|---|---|
| SOA | Start of Authority( acte de naissance) | Paramètres de la zone |
| A | Adresse IP d'u hote | Entier sur 32 bits |
| MX | Relais de messagerie | Domaine prenant le courrier |
| NS | nom de serveur | Nom d'un serveur pour ce domaine |
| CNAME | Nom de canonique | Nom de domaine |
| PTR | Pointeur | Alias pour une adresse IP |
| FO | Description de l'hote | UC et Système d'exploitation en ASCII |
| TXT | Texte | Texte ASCII non interprété |

# DNS : configuration (1)

Configuration du serveur pour le domaine

```
 IN      SOA     pegase.igh.cnrs.fr. postmaster.igh.cnrs.fr. (
                           2000010546  ; YYYYMMDDSS Serial
                           36000       ; Refresh every 10 hours
                           3600        ; Retry after 1 heure
                           604800      ; Expire after 7 jours
                           172800      ; MINimum TTL is 2 jours
                           )
;    Define name servers
igh.cnrs.fr.     IN      NS      pegase.igh.cnrs.fr.
IN      NS      lirmm.lirmm.fr.
;    Define MX for all domain
igh.cnrs.fr.     IN      MX  10  pegase.igh.cnrs.fr.
```

# DNS : configuration (2)

## Champs « A », « MX » et « CNAME »

```
localhost       IN  A           127.0.0.1

pegase          IN  A           195.83.84.20
                IN  MX      10  pegase.igh.cnrs.fr.
mail                    IN  CNAME       pegase.igh.cnrs.fr.
fmf                     IN  CNAME       pegase.igh.cnrs.fr.
genopole                IN  CNAME       pegase.igh.cnrs.fr.

ftp                     IN  CNAME       crystal.igh.cnrs.fr.
```

# DNS : types de requêtes

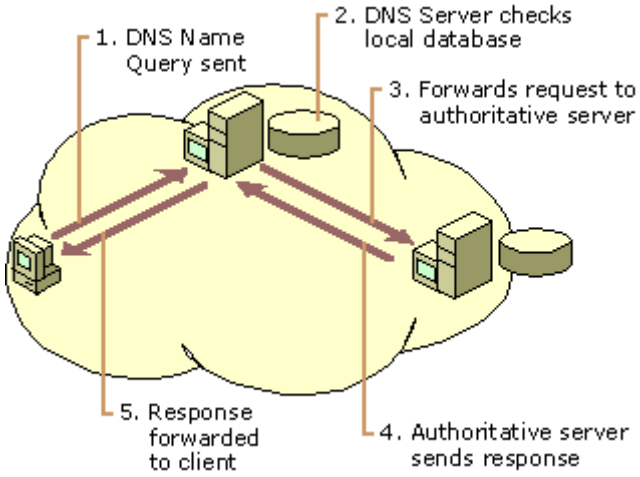
## Requêtes itératives :

Le serveur retourne soit la bonne réponse soit une référence vers un autre serveur de nom à contacter

Les serveurs non récursifs répondent seulement aux requêtes pour les zones pour lesquels ils font "autorité". Ils ne conservent pas d'information en cache !

## Requêtes récursives :

Les serveurs récursifs ont obligation de donner réponse au client. Ils questionnent à leur tour d'autres serveurs jusqu'à obtention de la réponse (ou erreur)...leur cache se remplit



# Commandes utiles pour DNS

## Dig : requêtes normales

```
$ dig +trace fr ns
$ dig +trace cnrs.fr ns
$ dig +trace igh.cnrs.fr ns

$ dig igh.cnrs.fr ns
$ dig ipnl.in2p3.fr mx
$ dig lyoinfo.in2p3.fr a
```

## Dig : requêtes inverses

```
$ dig +trace 195.in-addr.arpa ns
$ dig +trace 83.195.in-addr.arpa ns
$ dig +trace 84.83.195.in-addr.arpa ns
```

# SMTP (Rfc 821 & Rfc 822)

Définit :

L ’adresse électronique (RFC 822) :

Login@machine.Domaine ou prenom.Nom@domaine

Pas de différence entre minuscules et majuscules

Pas d ’accents, ni de caractères non imprimables.

Les caractères . et - autorisés. Le caractère _ non préconisé

Le format du message (entête et enveloppe)

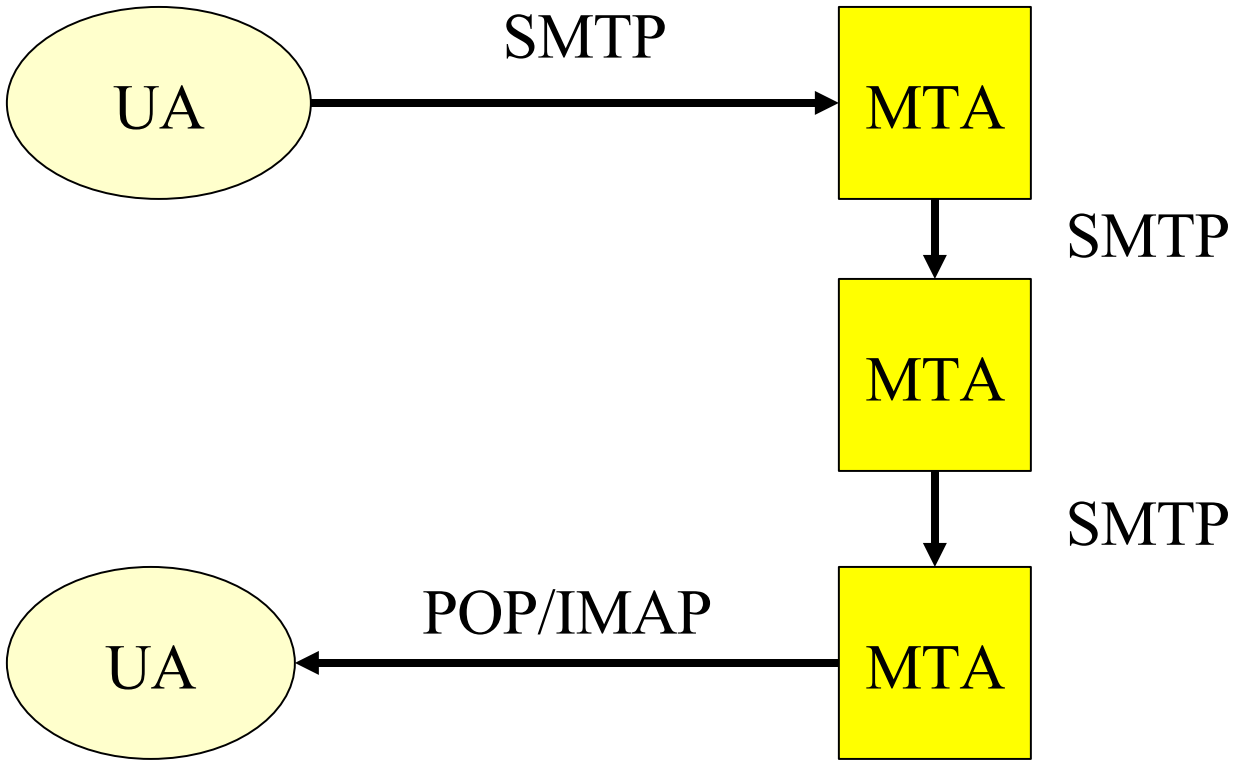
les notions suivantes :

UA (User agent) : votre logiciel de messagerie (Eudora, MailDrop, pine...)

MTA (Mail Transport Agent) : Un agent de transport reçoit un message et une direction, et l'achemine à l'endroit indiqué. Ne prends pas de décision sur le routage. Un agent de transport de messages est spécialisé pour un type de transmission particulier (local, UUCP…)

MRA (Mail Routing Agent) : Un agent de routage reçoit un message. En fonction de l'adresse du destinataire, il décide de faire appel à un agent de transport de messages, dont le but est d'acheminer le message dans la direction du destinataire.

# Courrier électronique



# SMTP : Commandes / réponses

Commandes :
HELO/EHLO (identifie le client)

MAIL FROM (adresse de l ’expéditeur)

RCPT TO (adresse du destinataire)

DATA (introduit le message)

réponses
2xx accepté

3xx attentes de données

4xx refusé (erreur temporaire)

5xx refusé (erreur permanente)

# SMTP : En tête d ’un message (1)

Voir la RFC 2076 : Common Internet Message Headers

**From** : Identité de l'expéditeur (la personne qui a souhaité que le message soit envoyé), placée par l'UA de l'émetteur.

**To** : Destinataires principaux du message, spécifiés par l'expéditeur à l'aide de son UA

**Cc** (carbon copy) : Destinataires auxiliaires du message, spécifiés par l'expéditeur à l'aide de son UA ;

**Bcc** (blind carbon copy) : Destinataires auxiliaires, spécifiés par l'expéditeur à l'aide de son UA. Ce champ n'est pas transmis aux destinataires spécifiés par To et Cc

**Date** : Date d'expédition, placée par l'UA de l'expéditeur

**Subject** : Sujet du message, spécifié par l'expéditeur à l'aide de son UA ;

**Reply-To** : Adresse de retour, placée par l'expéditeur, utilisée par le destinataire pour les réponses. Si ce champ n'est pas spécifié, From est pris par défaut ;

**Received** : Ajouté par chaque agent de routage le long du chemin emprunté par le message pour signer et tracer ce chemin en cas de problème.

**Return-Path** : Ajouté lors de la remise physique du message, c'est-à-dire lors du dépôt dans la boîte aux lettres finale par le dernier agent de transport, pour identifier le routage vers l'expéditeur

**Sender** : Identité de l'expéditeur réel : la personne (personne physique ou processus) qui a composé et envoyé le message, et qui reçoit les messages d'erreur liés au routage du message.

# SMTP : En tête d ’un message (2)

**Message-Id** : Identificateur unique du message, placé par le premier agent de routage, servant à référencer le message. In-Reply-To : En cas de réponse, référence au message original placée automatiquement par l'UA de l'expéditeur de la réponse ;

**References** : Identification de messages précédemment envoyés et cités en référence ;

**Comments** : Commentaires (aucune utilité pour le routage), spécifiés par l'expéditeur

**Encrypted** : Indique que le message est chiffré et spécifie la méthode utilisée

**X-???** : Les champs commençant par X- ne sont pas définis et sont réservés pour les extensions non encore standardisées ou pour des champs laissés aux utilisateurs.

**Resent-???** : messages par un utilisateur si son UA le lui permet.
Exemples (Resent-From, Resent-Reply-To, Resent-To, Resent-Cc…)

Le standard MIME définit également des champs d'en-tête :
**Mime-Version** : 1.0

**Content-Type** : text/plain; charset=« iso-8859-1 »

**Return-Receipt-To** : Pas ajouté par sendmail, mais par l'expéditeur du message. Lorsque sendmail réalise la remise physique du message et que ce champ est présent, il envoie un message de confirmation de remise (ce qui ne doit pas être confondu avec un accusé de réception) à l'adresse indiquée. Fortement déconseillée pour deux raisons essentielles :
il n'est reconnu que par sendmail, c'est-à-dire par une seule implémentation

il peut générer des boucles de courriers lorsqu'une liste de diffusion est en jeu.

**Precedence** : Ce champ donne la priorité devant être affectée au traitement du message.

# Fiabilité du courrier électronique

Pas fiable :

- Qui l ’a envoyé ? (adresse source)
- Quand est-il arrivé ?
- comment savoir si le correspondant l ’a reçu ?
- comment savoir si le correspondant l ’a lu ?

Erreurs fréquentes :

- « Host unknown » : la partie domaine (APRÈS @) est invalide !
- « User unknown » : la partie utilisateur (AVANT @) est invalide !
- erreur sur le serveur mail qu ’utilise le correspondant (config, disque…)

Transite en clair sur le réseau, stocké en clair sur les serveurs :

- Utiliser le chiffrement pour les messages sensibles / confidenciels
- Utiliser le SSMTP pour chiffrer le transport des messages entre le UA et le MTA ou entre MTA

# FTP (RFC959)

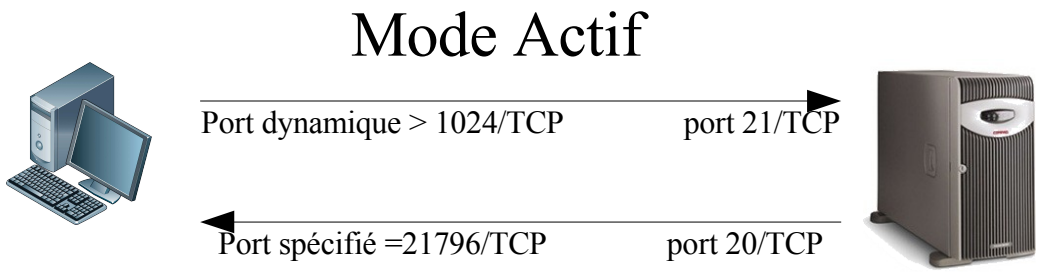
File Tranfer Protocol : Application pour transférer des fichiers

Utilise 2 connexions TCP :

1 canal de contrôle (commandes et réponses) : *port 21*

1 canal de données : cette connexion est ouverte puis fermée à chaque transfert

Modes actif / passif



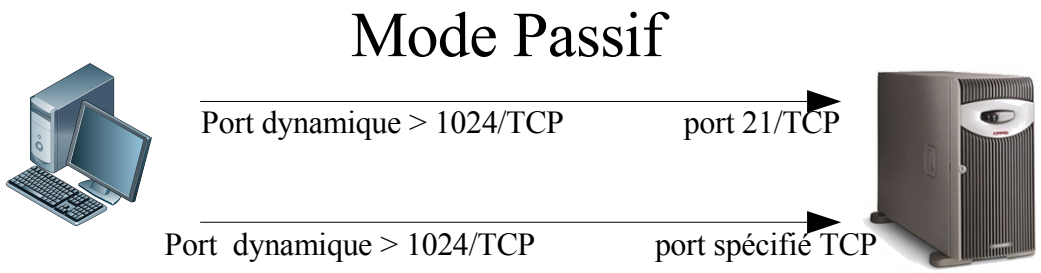

Le client FTP se connecte sur le port des commandes (21).

Le client FTP envoie la commande PORT N au serveur Ftp.

Exemple : PORT 192,168,0,186,85,36

Le serveur FTP initialise une connexion sur le port N (85*256+36=21796) du client pour engager un transfert.

# FTP : mode passif




Le client FTP se connecte sur le port des commandes (21).

Le client FTP envoie la commande PASV au serveur.

Le serveur FTP envoie la commande PORT N au client. Exemple :

```
PASV
227 Entering Passive Mode (192.168.0.186,215,205).
```

Le client FTP initialise une connexion sur le port N (215*256+205=55245) du serveur FTP pour engager un transfert. Exemple :

```
LIST
Connect socket #564 to 192.168.0.186, port 55245...
```

# FTP (3)

La réponse du serveur :

Précédée d'un nombre de 3 caractères :

1er chiffre permet de savoir à quoi se rapporte la réponse

1XX :

la commande commence a être exécutée, il y aura une autre réponse.

Ex :150 Opening data connection for ... (message en début de transfert)

2XX :

La commande a été exécutée avec succès.

Vous pouvez envoyer une autre commande.

Ex: 226 Transfer complete (message en fin de transfert)

5XX :

Commande non acceptée.

Ex: 550 fichier: No such file or directory en réponse à "get fichier"

# FTP (4)

Permet de transférer des fichiers

En mode "stream" ou "block" (Unix toujours en mode stream)

Texte (ASCII sous Unix)

Suite d'octets avec 7 bits significatifs.

Les fins de ligne, de page ... sont détectées et transformées si besoin pour être adaptées à la machine cible.

Il peut y avoir un transcodage: ASCII-EBCDIC

Binaire (Image)

Suite d'octets avec 8 bits significatifs.

Aucune transformation est apportée

Et les Macs ? La fameuse « ressource fork »

Mac Binary, Binhex, AppleDouble...

# FTP (5)

## Principales commandes

| | |
|---|---|
| help ou ? | liste des commandes |
| status | état des connexions |
| open | ouvrir une connexion (transparent) |
| user | entrer un nom d'utilisateur (transparent) |
| passwd | envoyer le mot de passe (transparent) |
| ls | fait un ls sur la machine distante |

Passer du mode binaire (**bin** : transfert de fichiers non texte) ou **ascii** (transfert de fichiers texte simple - sans mise en page-)

| | |
|---|---|
| cd | changer de répertoire sur la machine distante |
| get | rapatrier un fichier (ouvre une connexion TCP) |
| put | envoyer un fichier (ouvre une connexion TCP) |
| type | indiquer le type de fichier (ASCII ou binaire) |
| delete | effacer un fichier sur la machine distante |
| quit | fermer une connexion |

# TFTP (RFC 783)

- Utilise UDP
- Sans contrôle d'accès
- Utilisé pour charger le système dans des équipements sans mémoire non-volatile et sans disque
- Utilisé pour charger la configuration des routeurs

# SSH présentation

Connexions à distance : SecureShell

– SSH est un protocole, devant sécuriser les communications. En fait, SSH chiffre et compresse un tunnel de session qui sécurise les données transmises (permet d'éviter les sniffers réseaux)

Non seulement le mot de passe est chiffré lors de la connexion mais les informations circulant sur le réseau entre les deux machines le sont aussi.

SSH est composé d'un ensemble d'outils permettant des connexions sécurisées entre des machines. Ces outils ont pour but de remplacer les utilitaires de connexions classiques n'utilisant pas de chiffrage.

Remplace : rcp, rlogin, rsh, telnet(ftp par sftpen SSH V2)

# SSH : les versions

Terminologie de SSH :

C'est un protocole (2 versions) :

- Version 1
- Version 2

C'est aussi un produit :

- SSH Communications Security Inc(V1 et V2)
- Initialement développé par Tatu Ylönen
- OpenSSH du projet OpenBSD(V1 et V2)

C'est aussi une commande

# SSH : les composantes

- Fonctionnement sur le schéma d'un système client – serveur
- Les clients *ssh* demandent une ouverture de connexion au serveur *sshd*
- La boîte à outils SSH est généralement composée de :
  - Serveur :sshd
  - Clients : ssh, scp, sftp(ssh= slogin)
  - Des outils de gestion: ssh-add, ssh-agent, ssh-keygen
  - Les fichiers de configuration (OpenSSH) sont souvent dans:
    - Pour le serveur : /etc/ssh
    - Pour les clients : /etc/sshet $HOME/.ssh

# SSH : l'authentification

Une fois que la connexion sécurisée est mise en place entre le client et le serveur, le client doit s'identifier sur le serveur afin d'obtenir un droit d'accès.

Par mot de passe: Le client envoie un nom d'utilisateur et un mot de passe au serveur au travers de la communication sécurisé et le serveur vérifie si l'utilisateur concerné a accès à la machine et si le mot de passe fourni est valide

Par clés publiques : Si l'authentification par clé est choisie par le client, le serveur va créer un *challenge* et donner un accès au client si ce dernier parvient à déchiffrer le challenge avec sa clé privée

Par hôte de confiance : système équivalent aux systèmes utilisant rhostou hosts.equiv

# SSH et X11

X11 Forwarding:

relaye simplement toutes applications X11 à travers le canal chiffré

Ne pas configurer de variable *$DISPLAY*dans les scripts de connexion (.cshrc, .profile, .bashrcetc..), sshdoit remplir lui-même cette valeur

*Donc c'est plus simple que telnet !*

Il est nécessaire d'avoir un serveur X11 sur la machine du client et de le mettre en fonctionnement

Sous OpenSSH, serveurs et clients Unix sont configurés ***par défaut*** pour transmettre des données X11 dans le canal sécurisé

En cas de soucis, un test de fonctionnement peut être effectué en initiant la connexion avec l'option ***-X***(active l'X11 en cas de non configuration dans le fichier /etc/ssh/ssh_config: voir côté serveur):

**ssh–X login@machine**

# SSH : les autres services

Des services additionnels. Décrits dans la suite :

Agent SSH: utilisation en mode authentification forte :
Programme qui garde les clés privées en mémoire et qui fournit les services d'authentification aux autres clients SSH.

Agent Forwarding: utilisation en mode authentification forte :
Relaye les demandes d'authentification entre les différents clients-serveurs jusqu'au client initial (agent de mandatement)

Tunneling: permet de rediriger tout service TCP à travers un canal chiffré.

# SSH : coté serveur

## Le serveur : sshd

### Répertoire du serveur : /etc/ssh

Deux fichiers de configuration :

sshd_config : paramétrages lors des connexions vers le serveur local, ce fichier s'applique au démon sshd

ssh_config : paramétrages base et général du client ssh, ce fichier s'applique donc pour les commandes ssh, scp, sftp

Les fichiers des clés privées/publiques du serveur :

Clés compatible V1 (sshv1/ssf)

ssh_host_key(privée) , ssh_host_key.pub(publique)

Clés compatibles V2 (sshv2)

ssh_host_dsa_key(privée) , ssh_host_dsa_key.pub (publique)

ssh_host_rsa_key(privée) , ssh_host_rsa_key.pub (publique)

# SSH coté serveur : sshd_config

**PermitRootLogin**: autorise le compte *root* à se connecter

**StrictModes** yes: vérifie les permissions des fichiers et répertoires importants (accès au propriétaire uniquement)

**RSAAuthentication** yes : méthode d'authentification par RSA, ne fonctionne qu'avec la première version du protocole

**PubkeyAuthentication** yes:  méthode d'authentification forte (rsa ou dsa) en V2

**AuthorizedKeysFile** .ssh/authorized_keys: nom et localisation du fichier de clés publiques individuelles sur les hôtes locaux.

# SSH coté serveur : sshd_config

**PasswordAuthentication** yes: autorise la connexion par mot de passe

**PermitEmptyPasswords** no: interdit les connexions sans mot de passe

**X11Forwarding** yes: active le transfert X pour sshd

**X11DisplayOffset** 10: réservation d'un numéro d'affichage X11 afin d'éviter les collisions entre sshd et le vrai serveur X.

**X11UseLocalhost** yes: bind sur l'interface de la boucle locale, permet d'éviter les problèmes de proxy

**Subsystemsftp** /usr/lib/ssh/sftp-server: active le système de transfert de fichiers via sftp

# SSH coté serveur : sshd_config

**AllowUserslogin**: autorise le compte local et seulement ce compte à se connecter via ssh

**DenyUserslogin**: indique que ce compte ne pourra pas recevoir de connexions ssh

Possibilité de gérer l'hôte source : login@host

(AllowGroupsgroupou DenyGroupsgroup)

Si il y a plusieurs comptes ou groupes, la syntaxe est séparation des noms par des espaces.

# SSH coté serveur : ssh_config

**Host** *: spécifie les hôtes concernés par la configuration qui suit (adresse ipou nom DNS, * = toutes)

**ForwardAgent** yes: indique à l'agent que l'agent d'authentification doit être renvoyé vers la machine distante

**ForwardX11** yes: autorise la redirection du serveur graphique (possibilité de lancer des applications graphiques dans la session ssh)

**StrictHostKeyChecking** no: automatise la gestion des clés d'hôtes dans known_hosts. Si la clé n'existe pas, la connexion ne sera pas refusée, et sera rajoutée sans le demander à l'utilisateur (l'utilisateur ne verra pas les changements de clés du serveur)

**StrictHostKeyChecking** ask: demande à l'utilisateur s'il veut ajouter la clé dans le fichier known_hosts, puis permet la connexion, si la clé du serveur change, un message d'avertissement sera envoyé à l'utilisateur

# SSH coté serveur : ssh_config

**StrictHostKeyChecking** yes: vérifie que la clé publique de l'hôte distant existe sur l'hôte qui cherche à se connecter et ensuite autorise la demande de connexion. Une non connaissance de la clé ou un changement de clé du serveur implique alors un échec de la connexion ssh

Il faut donc fournir la clé publique du serveur distant à la configuration cliente qui cherche à se connecter

/etc/ssh/ssh_known_hosts: base de données des clés d'hôte

On ne peut donc pas se connecter la première fois sans avoir la clé installée

# SSH : l'authentification forte

Connexion par authentification forte :

Utilise un algorithme très puissant pour le chiffrement *(algorithme RSA/DSA)*

Ainsi chaque utilisateur possède son propre jeu de clés uniques (une clé privée = secrète, une clé publique = accessible par tous)

une nouvelle connexion nécessite l'installation de la clé privée sur le client et de la clé publique sur le serveur

le serveur va créer un *challenge* et donner un accès au client si ce dernier parvient à déchiffrer le challenge avec sa clé privée