# Sécurité Linux, environnement et contre-mesures

**Plan :**

- **Installation (*design, partitionnement, chiffrement*)**
- **Utilisateurs & groupes (authentification, propriété, spécificité)**
- **Système de fichiers & droits (rwx, pam,)**
- **Mise en place et sécurisation de services**

# Installation d’une Debian 10 virtualisée avec Virtualbox

**https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-10.1.0-amd64-netinst.iso**

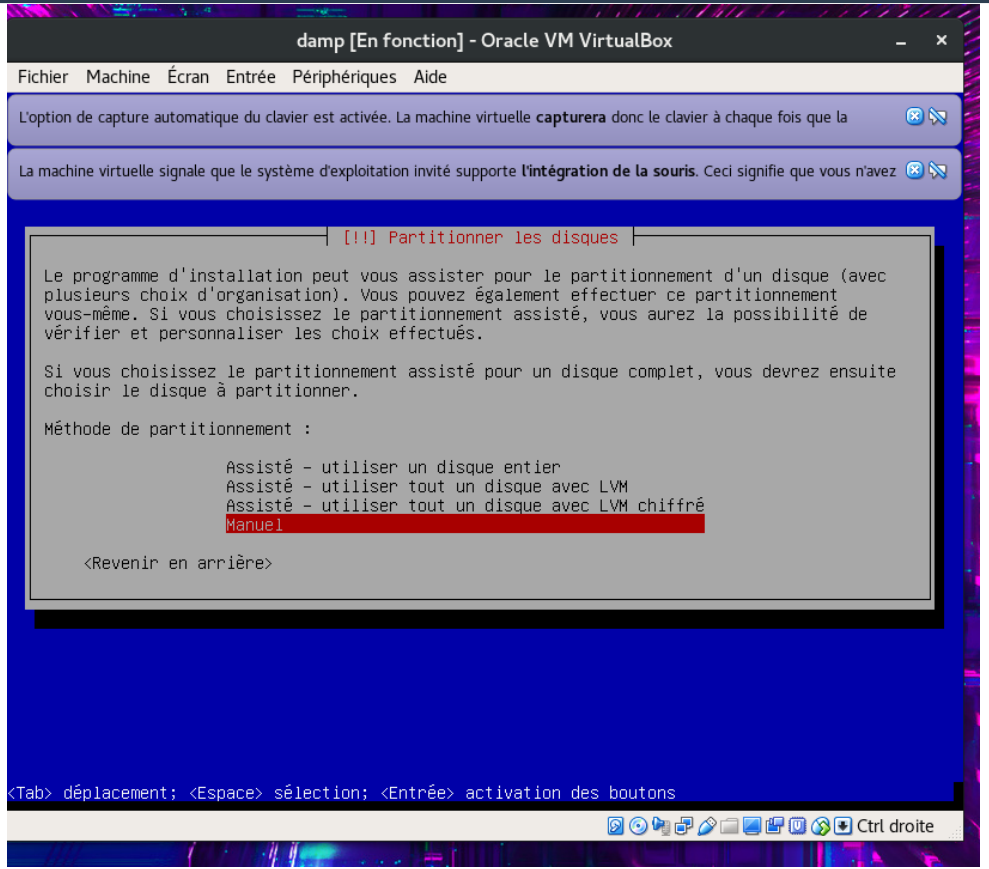
**Créer** nouvelle machine virtuelle (VM) avec les propriétés suivantes :

- 2048 Mo de RAM
- Disque virtuel de Type VDI 22 Go (ou plus)

Au démarrage, lui fournir l’ISO d’installation

# Installation d’une Debian 10 virtualisée avec Virtualbox

Suivre les options de base jusqu’à l’étape partitionnement



# Installation d’une Debian 10 (partitionnement)

Nous allons créer 4 partitions :

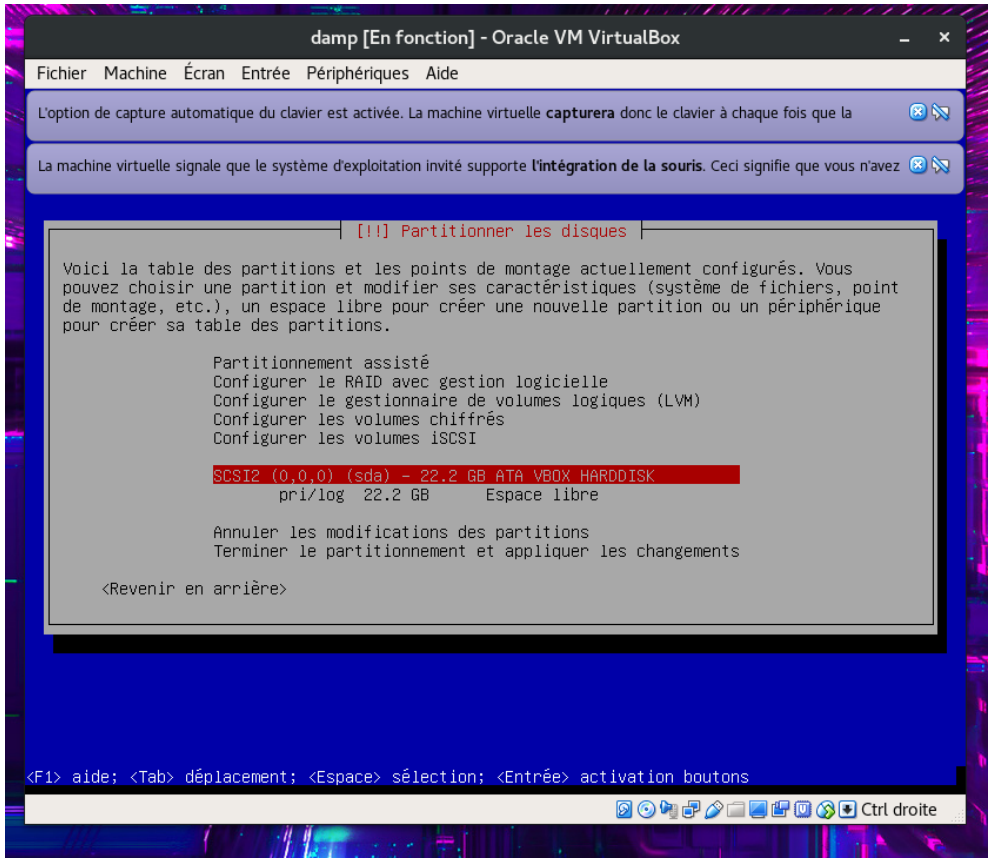
- / ↔ ext4 : 8Go
- /var ↔ ext4 : 4Go
- /home ↔ ext4 : 8Go
- Swap ↔ swap : 2Go

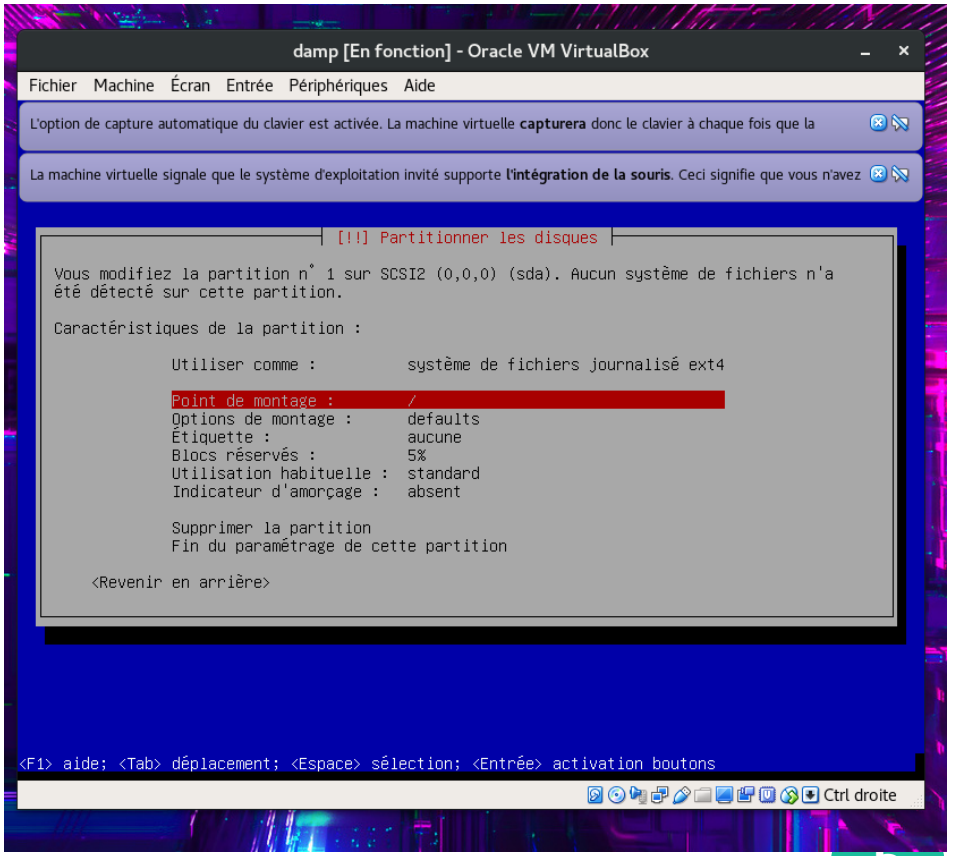
Pourquoi ne pas se contenter du partitionnement automatique ?

Données différentes = différentes

- Politiques de **rétention**
- Politiques de **chiffrement**
- Systèmes de **sauvegarde**
- éventuellement (type de device)

# Installation d’une Debian 10 (partitionnement)

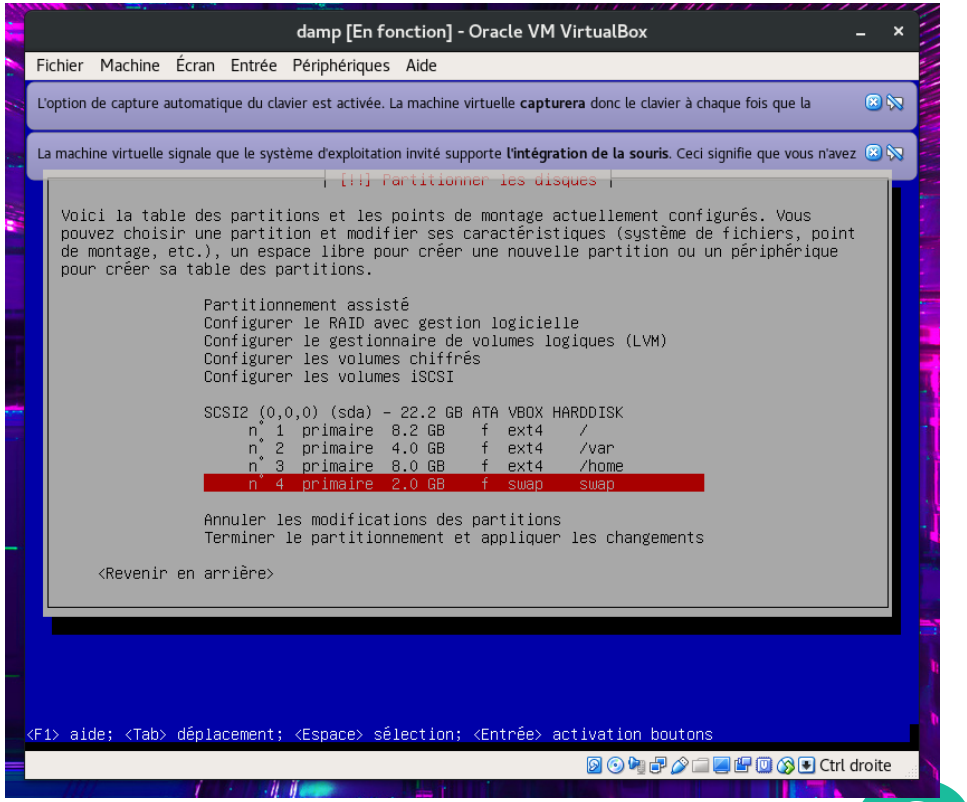


# Installation d’une Debian 10 (partitionnement)

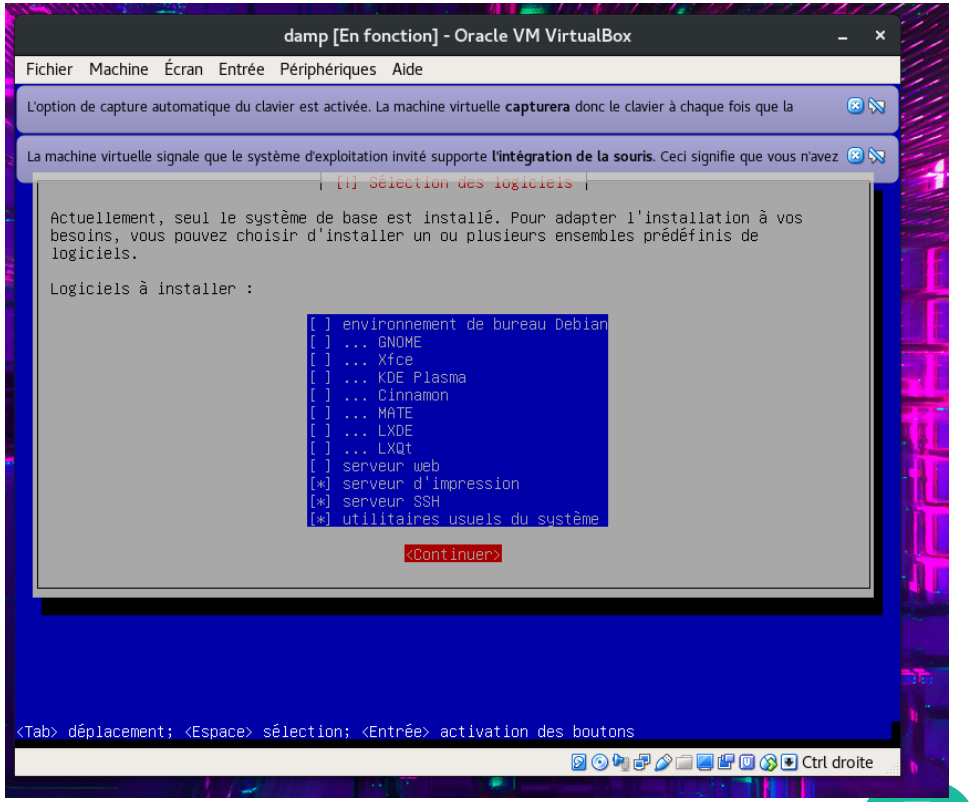
Voici à quoi doit ressembler votre partitionnement

*Terminer le partitionnement et appliquer les changements*



# Installation d’une Debian 10 (options environnementales)

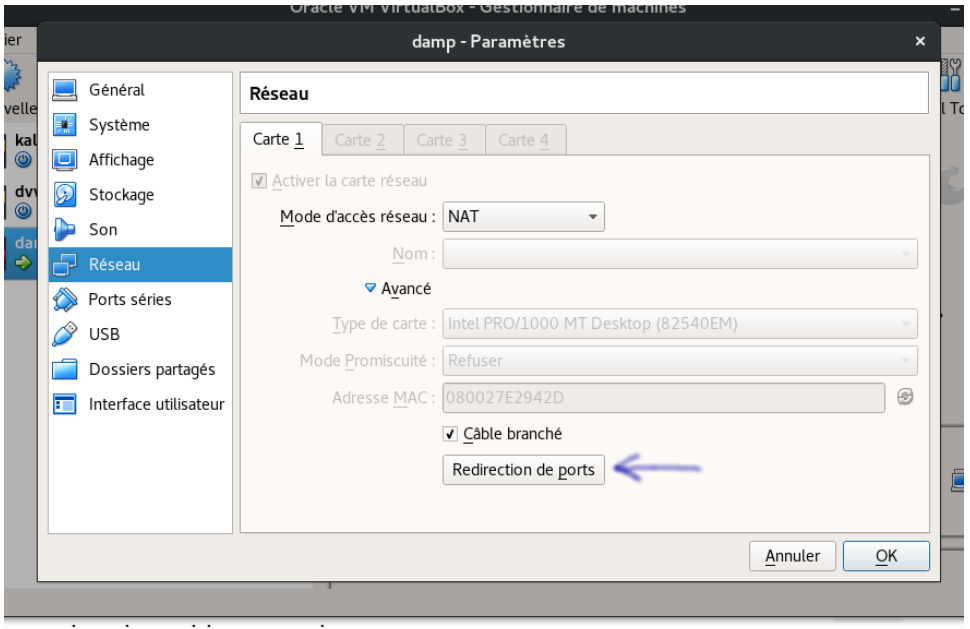
Pensons notre installation comme celle d’un serveur : nous n’avons **pas besoin d’environnement graphique**



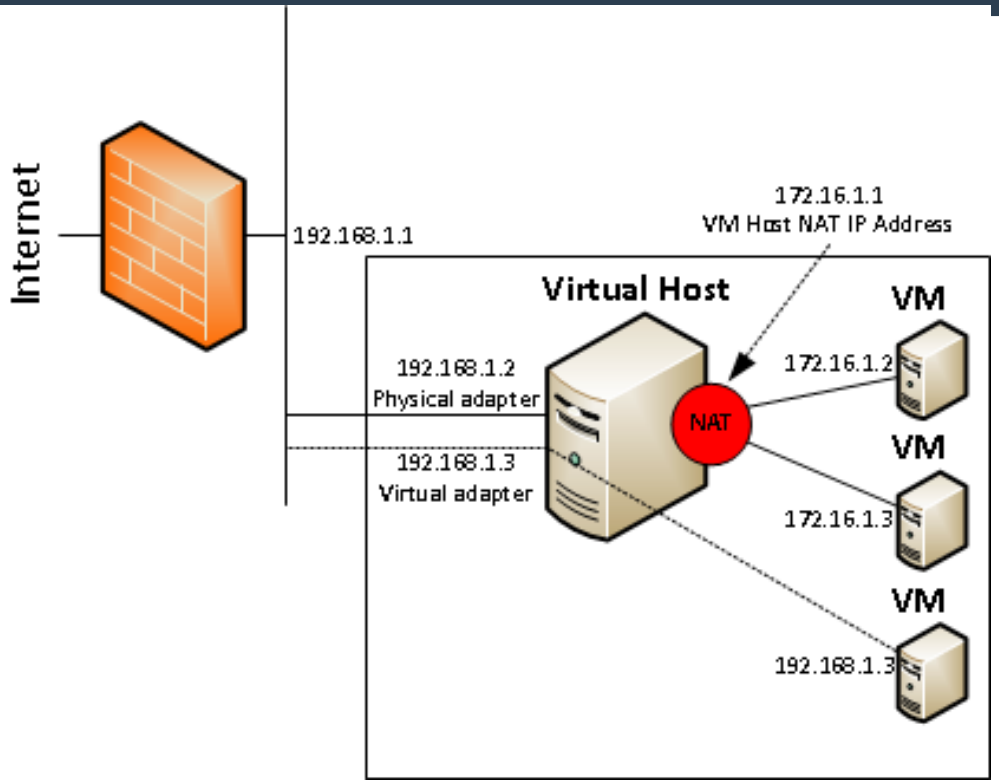
# Installation d’une Debian 10 virtualisée avec Virtualbox

**Côté Host cette fois.** Afin de faire propre, nous allons connecter notre VM au réseau en NAT.

Pour rendre ses services accessibles sans causer de conflit avec le **host**, nous allons mettre en place une redirection de ports



# Un point sur NAT vs Bridged

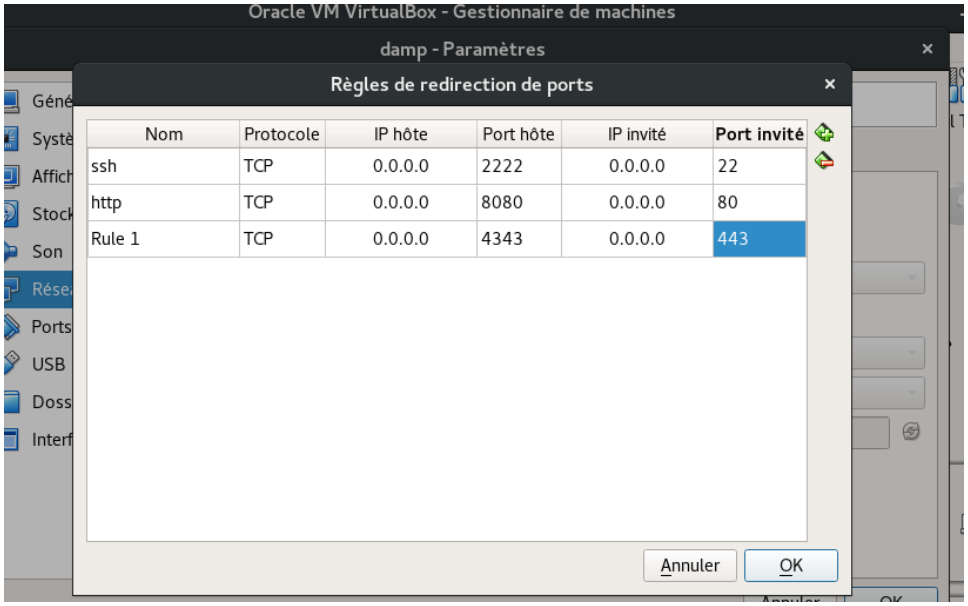



- **Network Adress Translation :** même IP pour **host** et **Guest**
- **Bridge :** le routeur attribut une ip locale au guest **(MAC - ARP)**

# Installation d’une Debian 10 virtualisée avec Virtualbox

**Côté Host cette fois.** Afin de faire propre, nous allons connecter notre VM au réseau en NAT.

Pour rendre ses services accessibles sans causer de conflit avec le **host**, nous allons mettre en place une redirection de port



# Promenons nous dans les shells

A l’aide du client ssh de votre choix, logger vous en utilisant l’utilisateur crée à l’installation

Remember : nous avons redirigé les ports

```
ssh votre_user@localhost -p 2222
```

Commençons par quelques questions existentielles

- Qui suis-je ?

  `whoami`

- Où suis-je ?

  `pwd`

- Quelle langue suis-je en train de parler ?

  `ps`

- A l’aide : `help`

# Promenons nous dans les shells

## Reconnaissance des lieux

- Noyaux et distribution

  `uname -a`

- Variable d'environnement

  `printenv`

- Ip locale (attention, le résultat va vous surprendre:-)

  `ip addr`

- Où sont les exécutables

  `echo $PATH`

## Puis-je utiliser un autre shell ?

Pour démarrer une session dans un autre shell, il suffit d'exécuter ce dernier :

`/usr/bin/sh`

Attention cela n'implique pas que les variables d'environnement changent nécessairement

Pour sortir : `exit`

# Bash : rappel des commandes de base

Qu’est-ce que le **Bourne-Again shell ?**

- Interpréteur en ligne de commande
- …..de type script → programmable \\/_

**NB** : BASH n’est qu’un shell parmi d’autres

- SH
- ZSH
- KASH

**Quelques commandes de base**

- **cd** chemin_vers_un_dossier/
- **ls** chemin_vers_un_dossier/
- **cat** fichier.txt
- **grep**
- **rm** chemin_vers_un_fichier
- **mkdir** un dossier
- Une_commande **>** un_fichier
- **touch** un fichier_a_creer
- |

# Exercice

A l’aide d’une de ces commandes, créer un fichier du nom de `authorized_keys` dans le dossier `.ssh` dossier de votre utilisateur

# Authentification SSH par clefs RSA : une bonne pratique

Pour nous connecter en SSH, nous avons tapé la commande suivie du mdp de l'utilisateur. Procéder systématiquement de la sorte présente **deux inconvénients** :

[***pratique***] besoin de taper le mdp à chaque fois (la vie est trop courte pour cela)

[***sécurité***] transmission systématique des credentials par le réseau (moyen moyen)

# Authentification SSH par clefs RSA : une bonne pratique

**Idéalement, il vaudrait mieux utiliser une clef cryptographique**

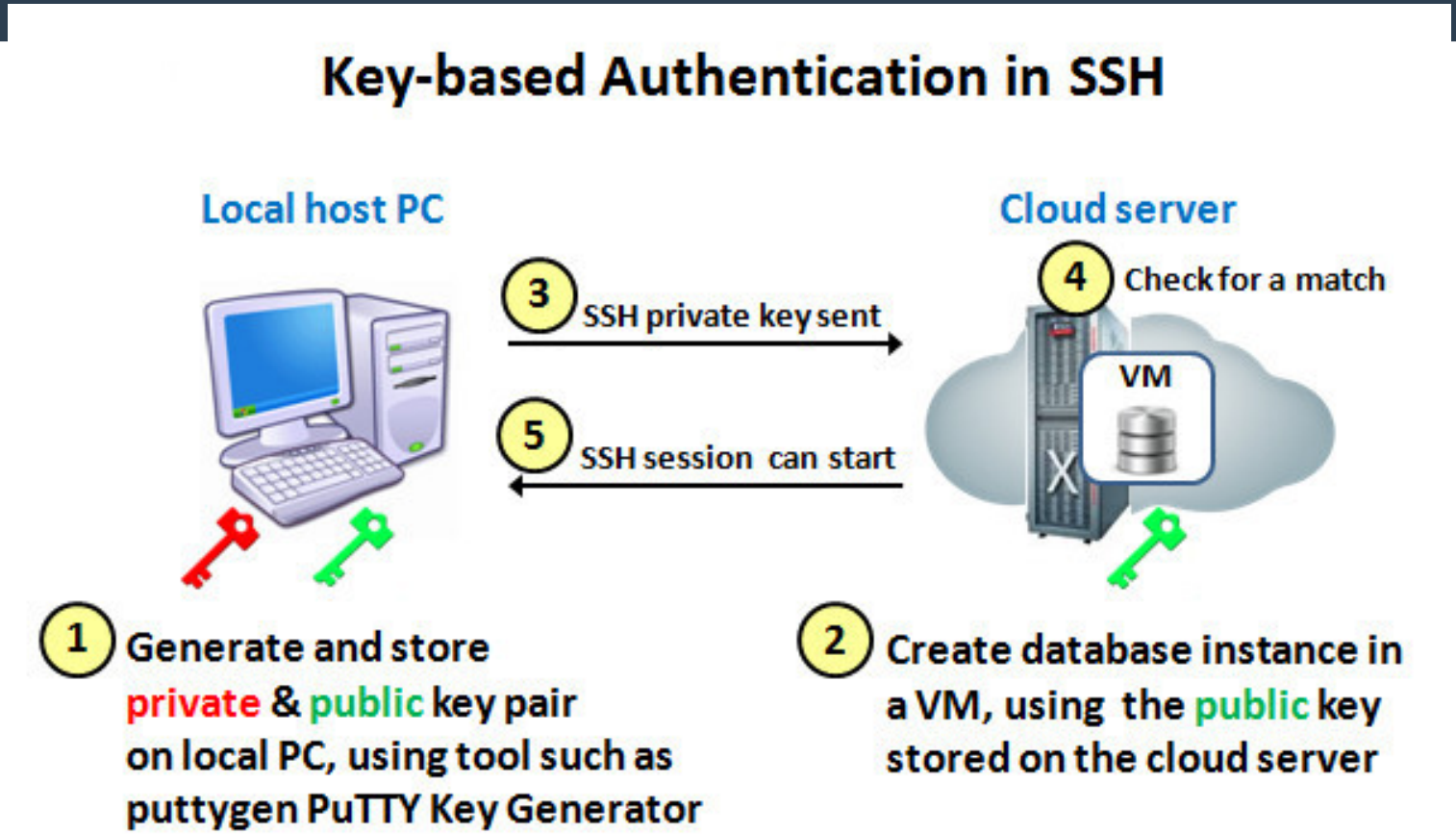
**- unique**

- qui assure le serveur de l'**identité** du client

- sans que celui n'ai besoin de transmettre par le réseau des informations que seul lui doit posséder (un mdp), par exemple

Système **Clef Privée** / **Clef Publique** (on parle de chiffrement asymétique RSA, DSA)

# Authentification SSH par clefs RSA : une bonne pratique



# Authentification SSH par clefs RSA : une bonne pratique

**En résumé :**

**[clef publique]** sera exportée sur chaque hôte sur lequel on souhaite pouvoir se connecter

**[clef privée]** permet de prouver son identité aux serveurs (« *Je jure et je prouve que c'est bien moi* »)

**[passphrase] :** Permet de sécuriser la clé privée sur la machine du client

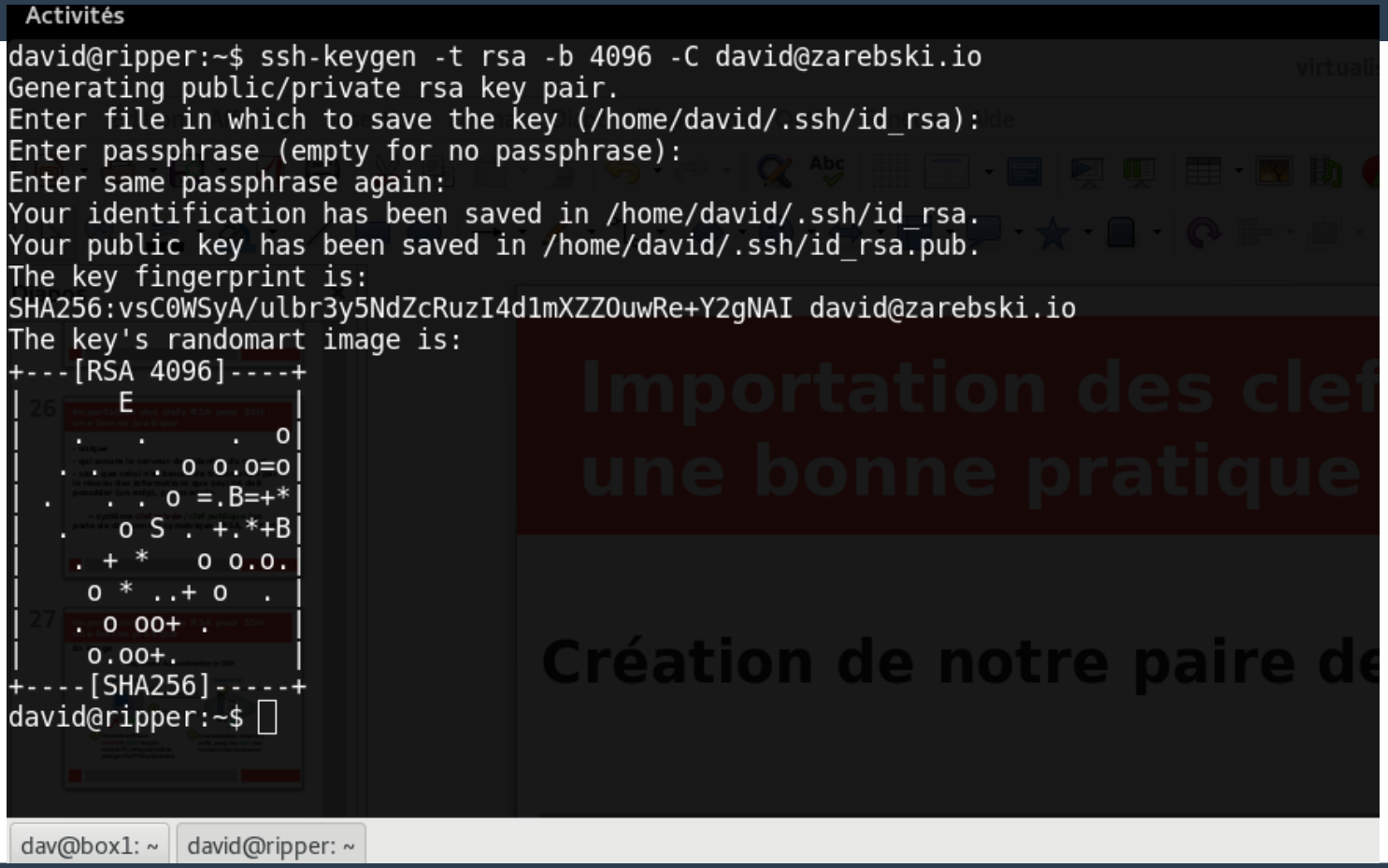
# Authentification SSH par clefs RSA : une bonne pratique

## Création de notre paire de clefs (côté client/host)

**ssh-keygen -t rsa -b 4096 -C**

**NB :** -C → commentaire (à des fins de clarté pour s'y retrouver avec les différentes clefs)

# Authentification SSH par clefs RSA : une bonne pratique



# Exercice : exporter la clef publique nouvellement générée

**Vous devez ajouter cette dernière au fichier authorized_keys** crée lors de l'exercice précédent. Pour cela, plusieurs solutions

- Ouvrir et éditer le fichier avec un éditeur du type de **nano** (user friendly) ou vi (moins user friendly)
- cat >> authorized_keys (puis coller votre clef publique)

**Vérifiez que cela a marché en**

- Vous dé-loggant
- Re-loggant sur votre serveur (vous ne devriez plus avoir à taper votre mot de passe)

# Envoi de notre clef au serveur (pour les braves)

```
cat ~/.ssh/id_rsa.pub | ssh -p 2222 "mkdir -p ~/.ssh && touch ~/.ssh/authorized_keys && chmod -R go= ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

(un peu barbare mais tout est fait d’un coup)

`chmod -R go ?` Nous y reviendrons plus tard lorsque nous aborderons les droits et les groupes

# Utilisateurs, Groupes, Privilèges

## Se logger comme root (pour ne plus jamais avoir à le faire)

**su**

Nous allons ajouter notre utilisateur au groupe des **sudoers**

```
sudo usermod -g sudo nom_de_votre_utilisateur
```

# De grands pouvoirs confèrent de grandes responsabilités

**MAJ** Maintenant que notre utilisateur est sudoer, il peut mettre à jour le système.

Sur Debian (et distributions dérivées), l'Advance Packaging Tool (**apt**) est le système de gestion des packets. A l'instar d'autres systèmes (**zypper**, **yum**, ), il repose sur un cache qu'il nous faut d'abord actualiser :

```
sudo apt-get update
```

avant de mettre à jour les paquets disposant de versions plus récentes

```
sudo apt-get upgrade
```

# Utilisateurs et groupes

Notre système ne dispose pour l'instant que de 2 utilisateurs

- **root**
- nom_de_l'utilisateur_créé_à_l'install

  ……. enfin, cela n'est pas exactement vrai  less /etc/passwd
- e.g. david:x:1000:1000:David Zarebski,,,:/home/david:/bin/bash
- Nom , mdp , uid , gid , ID Info , home ,  shell

Qui sont tout ces gens ?  cat /etc/group

nom_du_groupe:mot_de_passe:GID:liste_utilisateurs

# Utilisateurs et groupes

Tout utilisateur appartient à un groupe, fusse-t-il le group **nogroup**

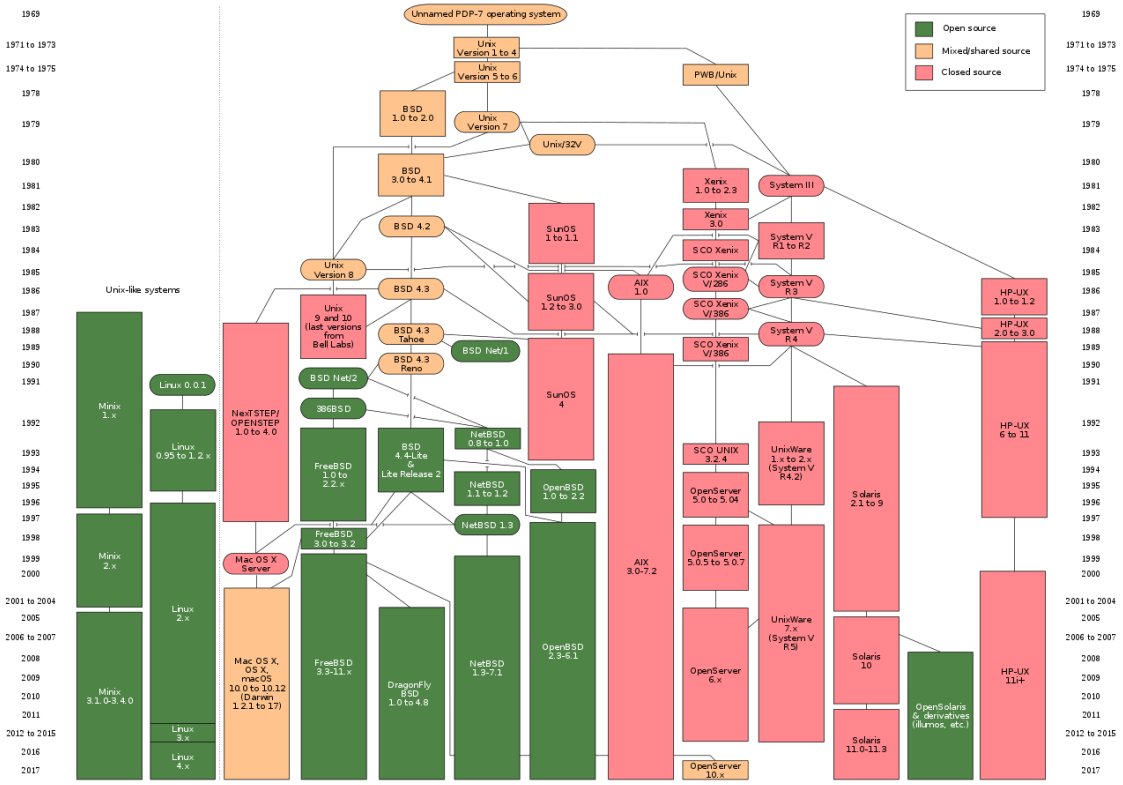
**EX** : Trouver une commande

- capable de lister l’ensemble des groupes auquel appartient votre utilisateur
- capable de lister l’ensemble des utilisateurs utilisant /bin/bash comme shell

# Utilisateurs et privilèges

Tout fichiers appartient à un utilisateur et un groupe.

**ls -la**



# Utilisateurs et privileges (r,w,x)

-rw------- 1 david david 8027 déc.  11 22:39 .bash_history

→ David peut lire et écrire

-rw-r--r-- 1 david david  220 nov.  17 13:41 .bash_logout

→ David peut lire et écrire, le groupe david et le reste du système ne peuvent que lire

drwx------ 3 david david 4096 nov.  20 14:25 .cache

→ Dossier : David peut lire, écrire et exécuter

drwxr-xr-x 2 david david 4096 nov.  17 14:08 damp

→ Dossier : David peut lire, écrire et exécuter, le groupe lire et exécuter, le reste du système, exécuter seulement

# Le changement des permission : chmod

Il est possible de changer les permissions sur un fichier à l'aide de la commande **chmod**.

Cette dernière est cependant plus facile à utiliser avec une notation des droits en format octal

| Octal Value | Read | Write | Execute |
|---|---|---|---|
| 7 | r | w | x |
| 6 | r | w | - |
| 5 | r | - | x |
| 4 | r | - | - |
| 3 | - | w | x |
| 2 | - | w | - |
| 1 | - | - | x |
| 0 | - | - | - |

chmod 750 chemin/fichier

- Owner : tous les droits
- Groupe : lecture exécution
- Le reste du monde : aucun droit

# Exercices

**EX1 :** Nous allons sécuriser SSH en restreignant les droits sur le fichier .ssh/authorized_keys . Modifer les droits de telle manière à ce que seul son propriétaire puisse le lire et l'écrire

-rw------- 1 david david  743 déc.  11 21:02 authorized_keys

**EX2 :** Créer un dossier contenant 10 fichiers et 2 dossiers (peu importe les noms). Editer les droits de telle manière que tous soient lisibles, modifiables et exécutables par le groupe *users* hormis 1 fichier et 1 dossier

# Création d’utilisateurs

Il serait possible de créer un nouvel utilisateur en éditant directement :

- /etc/passwd
- /etc/shadow
- /etc/group

Il existe cependant une alternative plus user friendly : useradd

- utilisez **useradd** pour créer un nouvel utilisateur
- allez regarder /etc/group, /etc/passwd et /etc/shadow

# Exercice en autonomie : renforcement de la sécurité du server ssh

Désactiver le root login (bien trop dangereux) en identifiant la ligne pertinente dans ce fichier de conf ;

`/etc/ssh/sshd_config`

Puis **redémarrez le service** avec

```
sudo systemctl restart sshd
```

**Mise en place d'un fail2ban**

l'idée en gros : si, pour un **service donné**, j'observe certain nombre d'**événements contigus** dans le temps, alors je **ban l'IP** à l'origine de ces requêtes pour **une durée déterminée**

Assurez-vous par la suite que cela marche en invitant quelqu'un à forcer l'authentification du serveur

# Chiffrement du HOME

# Des données sensibles

En tant qu'administrateur, votre **home** est amené à contenir de nombreuses informations cruciales :

- Historique de commande (bash_history)
- Code source de projets, variable d'environnement diverses

Dans le cas d'un **ordinateur personnel**, votre **home** ne contient rien de moins que :

- Votre historique de navigation
- Mdp enregistrés par votre navigateur

Les droits de lecture ne constitue qu'une réponse incomplète à cette problématique (pb de l'accès physique au disque)

# Chiffrement de fichiers sous Linux : deux grandes familles

Sous Linux, on peut chiffrer :

- La **partition** sur laquelle est montée /home
- Avec **dm-crypt** (par exemple)

**+** block-level, partition entière, on le fait une fois et plus besoin d'y penser

**-** « *tout le monde dans le même panier* »

- Les **dossiers** des utilisateurs contenus dans /home
- Avec **Ecryptfs** (par exemple)

**-** nécessaire de le faire pour chaque utilisateur...

**+** ...dont les données personnelles sont alors cryptografiquement protégées de tout le monde (y compris l'admin)

# Chiffrement de fichiers sous Linux : deux grandes familles

Sous Linux, on peut chiffrer :

- La **partition** sur laquelle est montée /home
- Avec **dm-crypt** (par exemple)

**-** pénible à mettre en place

- Les **dossiers** des utilisateurs contenus dans /home
- Avec **Ecryptfs** (par exemple)

**-** plus disponible dans les dépôts de Debian 10 en raison d'un bug avec **PAM** (rien de grave mais moins sûr)

# Chiffrement post install avec dm-crypt, du courage et un peu de bon sens

**Les étapes préliminaires** :

- Sauvegardez votre VM
- Se logger comme **root**
- Installer dm-crypt
  - apt-get install cryptsetup
- **Sauvegarder** les données de home
  - (ex : avec **rsync**)
- Delogger votre utilisateur
  - **pkill** -KILL -u {username}
- Identifier la partion **/home**
  - **fdisk** -l
- La démonter
  - umount /dev/sdaXX

# Chiffrement post install avec dm-crypt, du courage et un peu de bon sens

**Le chiffrement** (« pour de vrai »)

- silence, on chiffre (par simplicité pour la suite, utiliser le mdp de votre utilisateur en guise de **passphrase**)
- On peut éventuellement rajouter des clefs de déchiffrement de ce volume (en cas d'utilisateurs multiples)
- Enfin, on peut lister les clefs

- cryptsetup luksFormat -c aes-xts-plain64 -s 512 -h sha512 /dev/sdaXX
- cryptsetup luksAddKey /dev/sdaXX
- cryptsetup luksDump /dev/sdaXX

# Chiffrement post install avec dm-crypt, du courage et un peu de bon sens

## Déchiffrement automatique du volume au login

Il existe plusieurs solutions qui nécessitent que l'utilisateur saisisse sa **passphrase** en plus de son mdp. Il serait pas mal que le simple login déchiffre la partition

**Étape 1** : formater la partition déchiffrée en ext4

cryptsetup luksOpen /dev/sda3 cryptodisk

mkfs.ext4 /dev/mapper/cryptodisk

**Étape 2 :** modifier `/etc/fstab`

```
# /home was on /dev/sda3 during installation
/dev/mapper/cryptodisk	/home	ext4	defaults,noauto	0	2
```

# Chiffrement post install avec dm-crypt, du courage et un peu de bon sens

**Déchiffrement automatique du volume au login** modification du **PAM**

**Étape 3** : installation du module mount de pam

```
apt-get install libpam-mount
```

**Étape 4 :** ajout d'une règle dans `/etc/security/pam_mount.conf.xml`

```
# juste après le premier <pam_mount>
<volume user="*" mountpoint="/home" path="/dev/sda3" fstype="crypt" />
```

« *Mais qui c’est, cette PAM ?!?!* »

# Pluggable Authentication Modules

## Description canonique

Pluggable Authentication Modules (modules d'authentification enfichables, en abrégé PAM) est un **mécanisme** permettant d'intégrer différents schémas d'authentification de bas niveau dans une API de haut niveau, permettant de ce fait de rendre **indépendants du schéma les logiciels réclamant une authentification**.

## Comment vous pouvez le concevoir

Un genre d’API pour l’authentification d’utilisateurs sur le système

## Linux mais pas que

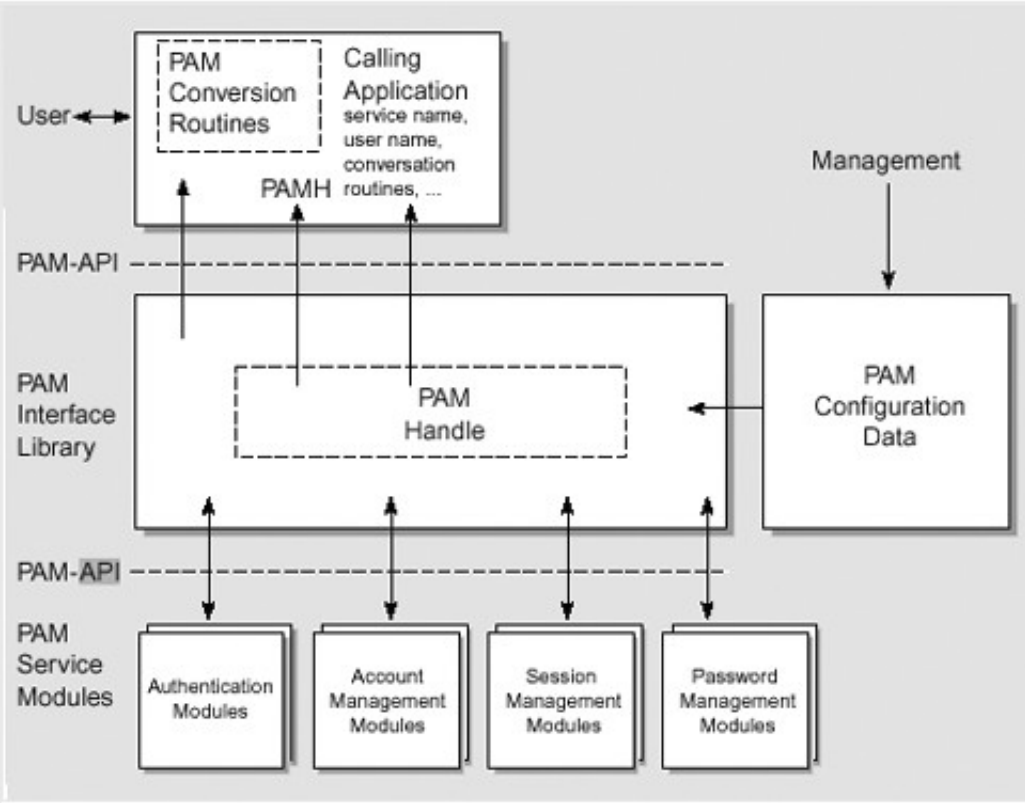
- Solaris
- OpenBSD
- Et autres UNIX

# Pluggable Authentication Modules

Pensez au fait suivant. Nous nous sommes connecté en ssh à la VM à l'aide de nos credentials, sommes arrivé dans une session bash sans avoir de configurer quoi que ce soit

Pensez-vous que **sshd** soit aller lire /etc/shadow, comparer les hashs, s'assurer de leur formats..... ?

Pensez-vous que toutes les interfaces d'administration des différents environnement graphique (kde, gnome, ) fasse de même ?



# Pluggable Authentication Modules

PAM repose sur quelques concepts fondamentaux

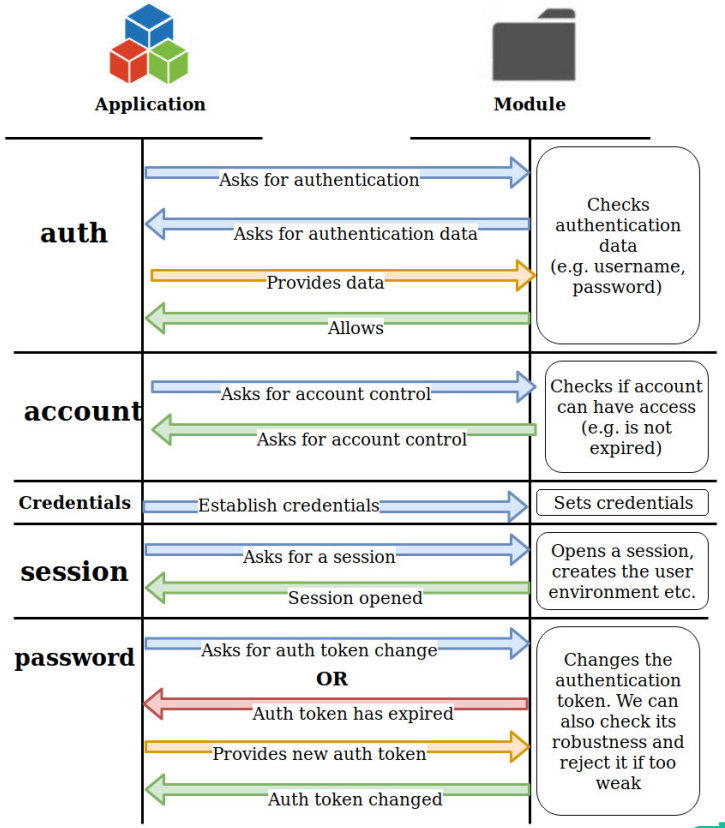
- **Application :** services utilisant PAM (aller voir voir /etc/pam.d)
- **Modules :** Il en existe quatre types. Il s'agit de petites librairies en C qui effectuent des taches standard comme :

  - charger des variables d'environnement

  - vérifier la valeur de certain parametres de con

  L'**empilement de ces modules** définit une politique de sécurité

  (on les trouve en /usr/lib/x86_64-linux-gnu/security/)

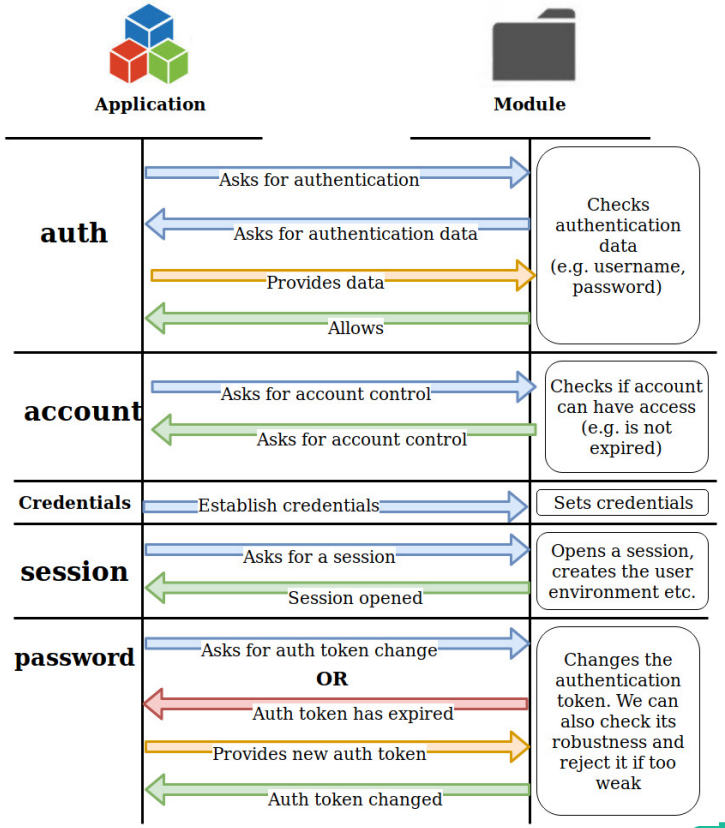


# Pluggable Authentication Modules

PAM repose sur quelques concepts fondamentaux

- **Drapeaux de contrôle :**
  - Required : le module doit réussir
  - Requisite : le module doit réussir et on ne lit pas les modules suivants en cas d'échec
  - Optimal : ne change rien à la suite
  - Sufficient : si correct, acceptation immédiate

(Vous l'aurez deviné l'**ordre d'appel des modules** a dont une importance)



# Pluggable Authentication Modules : un exemple

Intéressons nous a **sshd**

```
ldd /usr/sbin/sshd | grep libpam.so
```

**Analyse de son fichier de conf :** /etc/pam.d/sshd

- Hérite de l'authentification standard :

```
@include common-auth
```

- S'il existe un fichier en /etc/nologin, les utilisateurs non-root ne peuvent se logger

```
account    required     pam_nologin.so
```

- Récupère les variables d'environnement

```
session    required     pam_env.so
```

# Pluggable Authentication Modules : un exemple

**Exercice :** trouver le moyen d'afficher un message custom de votre choix à tout nouvel arrivant loggé via ssh

au cas où vous manqueriez d'inspiration https://asciiart.website/index.php

**NB : PAM n'est pas un deamon.** Vous n'avez pas besoin de recharger pam pour que les changements prennent effet, seulement **les services affectés par le changement** (dans le cas présent, sshd)

# let’s go real : récréons OVH (mais en mieux)

**Un scénario plus réaliste**

Votre serveur sert désormais de serveur de fichiers / serveur web pour de nombreux utilisateurs. Nous allons procéder aux précautions nécessaires pour :

- Protéger le serveur de ces derniers
- Les protéger les uns des autres
- Les protéger d’eux-mêmes (#DarwinAwards)

Etape préliminaire : installation

- Nginx

**Précision :** veillez à reporter l’ensemble des commandes nécessaires à la création de ce type d’utilisateur avec mdp par défaut dans un script

# let’s go real : récréons OVH (mais en mieux)

| Cahier des charges | Solutions techniques |
| --- | --- |
| • SFTP | • OpenSSH embarque déjà un server |
| • Shell restreint | • /bin/rbash ? () |
| • Vision limité du reste du système | • chroot \nn/ |
| • Dossier utilisateur invisible à un autre utilisateur | • 700 ;-) |
| • Mots de passe solides requis en cas de changement | • pam_cracklib |
| • Aucun access à **su** | |
| • Répertoire web lié dans /var/www | • Lien symbolique ln -s |

# let’s go real : récréons OVH (mais en mieux)

**Mais qu’est-ce qu’un chroot ?**

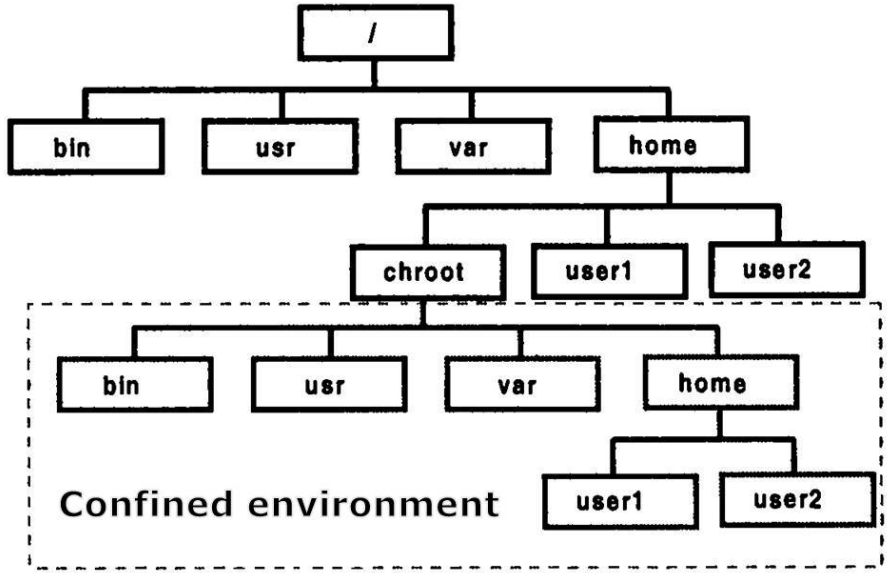
- Un environnement confiné

**Documentation**

**Comment chrooter les utilisateurs sshd ? Plusieurs alternatives :**

- libpam_chroot
- Fonctionnalité native d’openSSH>4.9+
- Recréer un environnement / dans le chroot

Nous allons privilégier cette dernière alternative (plus robuste)



# let’s go real : récréons OVH I) un groupe, une règle et un premier utilisateur

**création d’un groupe**

```
groupadd sftpusers
```

**édition de /etc/ssh/sshd_config**

```
Match Group sftpusers
   ChrootDirectory /home/%u
   AllowTcpForwarding no
   X11Forwarding no
```

**Création d’un utilisateur « test » sans domicile fixe**

```
useradd -M -g sftpusers -s /bin/bash test
```

```
passwd test
```

Et bien sûr, **on redémarre le service !!!!**

# let’s go real : récréons OVH III) recréation d’un environnement minimal

Le chroot doit contenir les éléments minimaux pour que l’utilisateur dispose d’un shell **bash** ainsi que quelque commandes (**ls**, **mkdir**, etc etc)

**Création du dossier chroot pour l’utilisateur test**

```
mkdir /home/test
```

(**root** doit en être propriétaire)

**Copie des binaires…..**

```
mkdir -p /home/test/bin
cp /bin/bash /home/test/bin
cp /bin/ls /home/test/bin
```

etc, etc

**…. et de leurs dépendances**

Pour chaque bin :

```
ldd /bin/le_bin_en_question
```

copiez les librairies en respectant l’arborescence source

# let’s go real : récréons OVH III) recréation d’un environnement minimal

Le chroot doit contenir les éléments minimaux pour que l’utilisateur dispose d’un shell **bash** ainsi que quelque commandes (**ls**, **mkdir**, etc etc)

Le chroot doit ressembler à quelque chose comme ça

(cette liste n’est pas exhaustive)

**/home/test/bin:**

bash  mkdir ls cd cp

**/home/test/lib/x86_64-linux-gnu/ :**

libselinux.so.1 libattr.so.1 libdl.so.2

**/home/test/lib64 :**

ld-linux-x86-64.so.2

# let’s go real : récréons OVH III) recréation d’un environnement minimal

## Création de /dev

A minima, l’utilisateur aura besoin d’un terminal :

```
mkdir -p /home/test/dev/
cd /home/test/dev/
mknod -m 666 null c 1 3
mknod -m 666 tty c 5 0
mknod -m 666 zero c 1 5
mknod -m 666 random c 1 8
```

## Création de /etc

A minima, l’environnement doit contenir /etc/passwd et /etc/group :

```
mkdir /home/test/etc
cp -vf /etc/{passwd,group} /home/test/etc/
```

# let’s go real : récréons OVH (mais en mieux)

Essayez maintenant de vous logger avec l’utilisateur « test »

Un problème ? Allez voir les logs

```
cat /var/log/auth.log | grep test
```

# let’s go real : récréons OVH IV) fignolage

**Création d’un /home** *Même les utilisateurs chrooté ont le droit à un home*

```
mkdir -p /home/test/home/test
```

```
chown test /home/test/home/test
```

**Création d’un www**

```
mkdir /home/test/www
```

```
cd /home/test/www
```

```
wget https://gist.githubusercontent.com/pitch-gist/2999707/raw/8dcc32cd374a01f53cec0a10cf558b30035672d4/gistfile1.html
```

```
mv gistfile1.html index.html
```

```
chown test /home/test/www
```

# let’s go real : récréons OVH V) Nginx


**2 étapes :**

- ln -s  /home/test/www/ /var/www/html/test
- Édition de /etc/nginx/sites-available/default

```
server {
    listen 80;
    listen [::]:80;
    server_name test.com;
    root /var/www/html/test;
    index index.html;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

# Exercice : durcissons la politique de mots de passe

Ce coup-ci, je vous laisse gérer :

https://www.cyberciti.biz/faq/securing-passwords-libpam-cracklib-on-debian-ubuntu-linux/

# Un point sur SELinux

# Networking

# Quelques outils réseaux indispensables de Linux

**ping** → transparent

**nmap** → découverte de ports et service

**dig** → un équivalent de **nslookup** (résolution de nom de domaine)

# Sécurisation d’un serveur Apache
# À l’aide de **mod_security**

# Installation et activation du module

`apt-get install libapache2-mod-security2`

`a2enmod security2`

Regardez ensuite ce que nous dit `/etc/apache2/mods-enabled/security2.conf` et tirez en les conclusions qui s’imposent

# Paramétrage de mod_security

**/etc/modsecurity/modsecurity.conf couvre la plupart des besoins basiques, hormis celui de bloquer les attaques plutôt que les logger (*Detection Only*). Nous allons remédier à cela :**

**SecRuleEngine On**

# Activation des règles additionnelles d’OWASP

```
  GNU nano 2.7.4

<IfModule security2_module>
        # Default Debian dir for modsecurity's persistent data
        SecDataDir /var/cache/modsecurity

        # Include all the *.conf files in /etc/modsecurity.
        # Keeping your local configuration in that directory
        # will allow for an easy upgrade of THIS file and
        # make your life easier
        IncludeOptional /etc/modsecurity/*.conf

        # Include OWASP ModSecurity CRS rules if installed
        IncludeOptional /usr/share/modsecurity-crs/owasp-crs.load
</IfModule>
```

**Redémarrez apache pour vous assurer que tout roule**