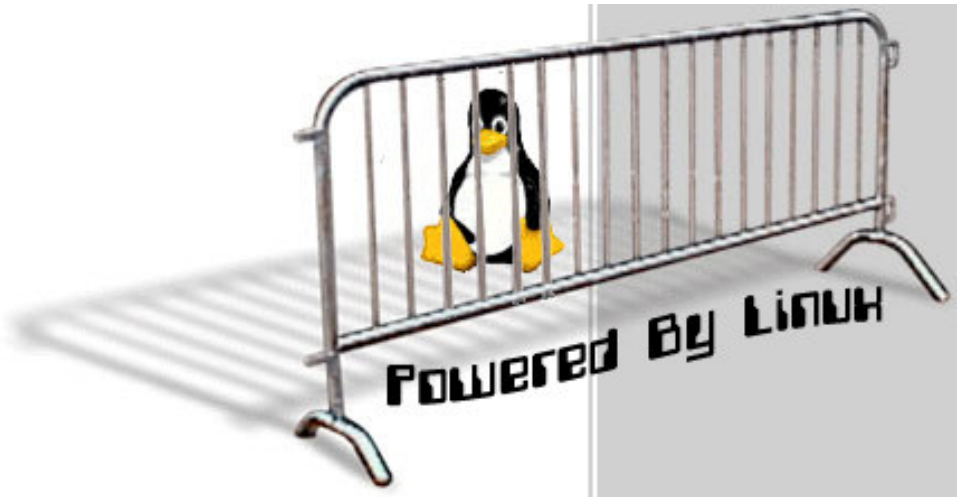
Julien Vehent

BTS Informatique de gestion – Option ARLE

Sécuriser un serveur Linux Red Hat 9.0



Serveur Linux Red Hat 9.0

Récapitulatif des compétences mises en œuvres :

- ✓ *C21 Installer et configurer un micro-ordinateur*
- ✓ *C23 Installer et configurer un dispositif de sécurité*
- ✓ *C31 Assurer les fonctions de bases de l'administrateur réseau*
- ✓ *C34 Surveiller et optimiser le trafic sur le réseau*

1. Introduction

Assurer la sécurité du réseau informatique est indispensable au bon fonctionnement de l'entreprise. Un administrateur système et réseau doit être capable d'identifier et de corriger d'éventuelle faille afin de limiter un maximum les risques de compromissions du réseau.

Ici, nous nous intéresserons à un serveur Linux Red Hat 9 qui assure les fonctions de DNS, DHCP, Samba et Postfix. Nous verrons comment sécuriser les accès depuis le réseau et comment configurer le serveur pour qu'il assure sa fonction sans service inutile.

2. Identification des fonctions du serveur

Les fonctions du serveur linux seront :

- la définition automatique des adresses IP des postes du réseau via le protocole DHCP.
- Le partage de fichiers sur les postes en Windows 2000 pro via l'outil Samba.
- Le service de messagerie électronique avec Postfix (protocoles STMP & POP3).
- La résolution des noms FQDN avec DNS (comprend le protocole RNDC également).
- L'accès à distance avec SSH et Telnet.

On peut donc déjà identifier les ports qui devront être ouverts.

TCP	**UDP**
22 & 23 / SSH & Telnet	53 / DNS
25 / SMTP	67 / DHCP
110 / POP3	137 & 138 / Samba
953 / RNDC	

En utilisant l'outil Nmap (commande : *nmap localhost*), nous sommes en mesure de vérifier quels sont les ports ouverts sur notre serveur.

3. Configuration du lancement des services.

Par défaut, de nombreux services sont lancés lors de l'installation de Linux Red Hat 9. Cela pose deux problèmes :

- Le fonctionnement de services inutilisés consomme des ressources machines qui pourraient être dédiées à d'autres tâches.

- Le risque de compromissions de notre serveur augmente pour chaque service en fonctionnement. En effet, de nombreux services comportent des failles qui peuvent être exploitées par un pirate (buffer overflow, etc…).

Nous allons donc trier les services pour ne conserver que ceux qui sont indispensables au fonctionnement du serveur.

La commande *chkconfig --list* permet de lister les services qui seront lancés au démarrage. Les services que nous conserverons sont les suivants :

Syslog	gestion des logs
Network	gestion du réseau
Random	génération aléatoire
Rawdevices	partition spécifique pour sgbdd
Apmd	advanced power management
Keytable	gestion du clavier
Kudzu	détection du matériel
Sshd	Serveur SSH
Crond	gestion des tâches programmées
Smb	démon samba
Dhcpd	démon dhcp
Named	démon DNS
Postfix	démon messagerie
Xinetd	méta-démon de lancement de services

Service dépendants de Xinetd :

Telnet	Démon de connexion à distance
Ipop3	Démon de récupération de mails

Et, si on a besoin du serveur graphique :

Xfs	serveur de polices de caractères

La commande *chkconfig –level 2345 NOM_DU_SERVICE off* nous permet de désactiver le lancement des services non désirés au démarrage du système.

4. Blocage des comptes inutiles

A l'installation du système, de nombreux comptes sont créés. Ils correspondent pour la plupart à des comptes d'applications et n'ont pas pour vocation d'être utilisés comme comptes de connexions. Nous allons donc conserver uniquement les comptes dont nous avons autorisé la connexion sur le serveur. Le shell de tous les autres doit pointer vers */sbin/nologin.*

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/sbin/nologin
shutdown:x:6:0:shutdown:/sbin:/sbin/nologin
halt:x:7:0:halt:/sbin:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin

[…]

xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
named:x:25:25:Named:/var/named:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
gdm:x:42:42::/var/gdm:/sbin/nologin
amanda:x:33:6:Amanda user:/var/lib/amanda:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
mailman:x:41:41:GNU Mailing List Manager:/var/mailman:/sbin/nologin
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/sbin/nologin
radvd:x:75:75:radvd user:/:/sbin/nologin
pierre:x:500:500::/home/pierre:/sbin/nologin
paul:x:501:501::/home/paul:/sbin/nologin
jacques:x:502:502::/home/jacques:/sbin/nologin
julien:x:503:503::/home/julien:/bin/bash
nel:x:504:504::/home/nel:/sbin/nologin
switcher:x:505:505::/home/switcher:/sbin/nologin
```

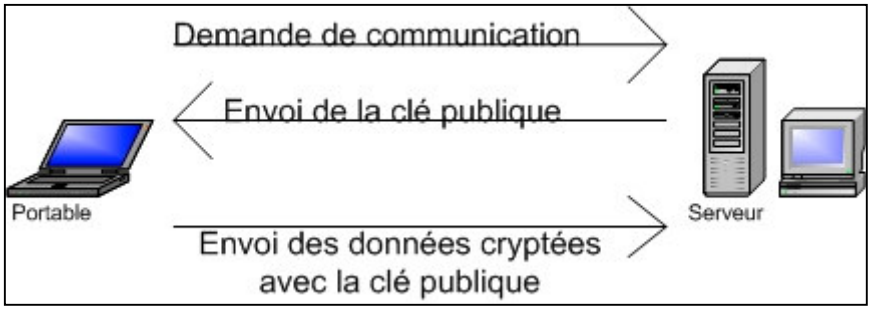
5. Le protocole SSH

Le protocole SSH est une alternative sécurisée au serveur Telnet classique. Comme ce dernier, il permet d'accéder à la console du serveur à distance. Le gros avantage de SSH est qu'il utilise l'algorithme de cryptage RSA (infrastructure de clé publique) alors que Telnet n'est absolument pas sécurisé.

5.1 L'infrastucture de clé publique.

Deux postes A et B veulent communiquer à travers le réseau. B est un serveur linux sur lequel le démon SSHD fonctionne. A est un client sur lequel le logiciel Putty est installé.

Au lancement du service SSHD sur B, deux clés sont générées. L'une sera privée et l'autre publique.
Seule la concaténation des clés privées et publiques permet le décodage.



Le poste A effectue une demande de connexion sur le poste B. Alors, B transmet à A sa clé publique.
Cette clé va permettre à A de chiffrer les données qu'il envoie à B. Et seul B, qui aura conservé la clé privée, pourra déchiffrer ces données.

5.2 Configuration de SSH

Nous allons utiliser la version 2 du protocole SSH qui, par rapport à la première version, corrige un grand nombre de faille.
SSH est installé dans */etc/ssh* et son fichier de configuration est *sshd_config.*

```
#Extrait du fichier de configuration de SSH

#force le fonctionnement en protocole 2
Protocol 2
#authentification par mot de passe
PasswordAuthentication yes
#interdit la possibilité de ne pas mettre de mot de passe
PermitEmptyPasswords no
#authentification RSA pour le protocole 2
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
#authentification de type Rhosts désactivée
RhostsAuthentication no
#ignore les fichiers utilisateurs .rhosts et .shosts
IgnoreRhosts yes
#authentification combinant RSA et Rhosts désactivée.
HostbasedAuthentication no      #pour le protocole 2
RhostsRSAAuthentication no      #pour le protocole 1
```

6. Telnet et SSH

Lors d'une ouverture de session distante entre deux machines avec l'outil TELNET, on « sniff » les trames qui passent par la carte réseau avec le logiciel Ethereal.

```
⊞ Frame 98 (281 bytes on wire, 281 bytes captured)
⊞ Ethernet II, Src: 00:50:8d:f1:ac:7d, Dst: 00:10:4b:e1:e4:bc
⊞ Internet Protocol, Src Addr: 192.168.1.99 (192.168.1.99), Dst Addr: 192.168.1.3 (192.168.1.3)
⊞ Transmission Control Protocol, Src Port: telnet (23), Dst Port: 1106 (1106), Seq: 168, Ack: 134, Len: 215
⊟ Telnet
    Data: \r\n
    Data: \r\n
    Data: ========================================================================\r\n
    Data: Bienvenue \205 Microsoft Telnet Server.\r\n
    Data: ========================================================================\r\n
    Data: C:\Documents and Settings\administrateur>
```

Sur la capture d'écran ci-dessus, on voit les données qui transitent en clair. Les transmissions ne sont pas sécurisées.

La même manipulation est effectuée entre les deux mêmes machines, mais cette fois c'est avec SSH que l'on ouvre une session distante.

```
 6 0.847169   192.168.1.3    192.168.1.99   DNS    Standard query PTR 99.1.168.192.in-addr.arpa
 7 0.848320   192.168.1.99   192.168.1.3    DNS    Standard query response PTR pcswitcher.mshome.net
 8 0.854476   192.168.1.3    192.168.1.99   SSH    Server Protocol: SSH-2.0-OpenSSH_3.4p1 Debian 1:3.4p1-1.woody.3
 9 0.865599   192.168.1.99   192.168.1.3    SSH    Client Protocol: SSH-2.0-PuTTY-Release-0.54
10 0.865800   192.168.1.3    192.168.1.99   TCP    22 > 4548 [ACK] Seq=48 Ack=28 Win=5840 Len=0
11 1.033707   192.168.1.3    192.168.1.99   SSHv2  Server: Key Exchange Init
12 1.055256   192.168.1.99   192.168.1.3    SSHv2  Client: Key Exchange Init
13 1.055398   192.168.1.3    192.168.1.99   TCP    22 > 4548 [ACK] Seq=592 Ack=516 Win=6432 Len=0
14 1.055295   192.168.1.99   192.168.1.3    SSHv2  Client: Diffie-Hellman Key Exchange Init
15 1.055459   192.168.1.3    192.168.1.99   TCP    22 > 4548 [ACK] Seq=592 Ack=532 Win=6432 Len=0
16 1.274972   192.168.1.3    192.168.1.99   SSHv2  Server: Diffie-Hellman Key Exchange Reply
17 1.383499   192.168.1.99   192.168.1.3    TCP    4548 > 22 [ACK] Seq=532 Ack=1016 Win=16505 Len=0
18 1.508364   192.168.1.99   192.168.1.3    SSHv2  Client: Diffie-Hellman GEX Init
19 1.508437   192.168.1.3    192.168.1.99   TCP    22 > 4548 [ACK] Seq=1016 Ack=948 Win=7504 Len=0
20 2.949799   192.168.1.3    192.168.1.99   SSHv2  Server: Diffie-Hellman GEX Reply
21 3.093227   192.168.1.99   192.168.1.3    TCP    4548 > 22 [ACK] Seq=948 Ack=1752 Win=17520 Len=0
22 7.000019   192.168.1.99   192.168.1.3    SSHv2  Client: New Keys
```

Tout d'abord, on voit ci-dessus l'échange des clés entre le serveur et le client. L'algorithme de calcul « Diffie-Hellman » est utilisé. Une fois que les deux machines possèdent la clé publique de leur homologue, les communications sont cryptées.

```
   102 15.460681   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   103 15.461695   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   104 15.462405   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   105 15.463529   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=132
   106 15.464111   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   107 15.465359   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   108 15.466261   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   109 15.467162   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   110 15.467970   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   111 15.468978   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   112 15.469680   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   113 15.470805   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   114 15.471499   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   115 15.472996   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   116 15.473544   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   117 15.474828   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   118 15.475694   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   119 15.476663   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   120 15.477414   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   121 15.478775   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   122 15.479559   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   123 15.480913   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=116
   124 15.481702   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
   125 15.516768   192.168.1.3    192.168.1.99   SSHv2  Encrypted response packet len=132
   126 15.517440   192.168.1.99   192.168.1.3    SSHv2  Encrypted request packet len=52
⊞ Frame 102 (106 bytes on wire, 106 bytes captured)
⊞ Ethernet II, Src: 00:50:8d:f1:ac:7d, Dst: 00:10:4b:e1:e4:bc
⊞ Internet Protocol, Src Addr: 192.168.1.99 (192.168.1.99), Dst Addr: 192.168.1.3 (192.168.1.3)
⊞ Transmission Control Protocol, Src Port: 4548 (4548), Dst Port: 22 (22), Seq: 2724, Ack: 3484, Len: 52
⊟ SSH Protocol
  ⊟ SSH Version 2
      Encrypted Packet: 7439A75AC8D2425EDA3DA8F7D5A25682...
```

La sécurité des transmissions est assurée, ces données ne sont pas compréhensibles.

7. Conclusion

La politique de sécurité que nous venons de mettre en place est une première étape vers la sécurisation totale d'un réseau. Ainsi il reste encore de nombreux points que nous aurions pu aborder tel que le filtrage des communications réseau via NetFilter, ou encore la sécurisation du fonctionnement des services (changement de racine d'exécution avec *chroot*, etc…).

De plus, il est important de surveiller constamment l'activité du réseau avec des outils comme Ethereal, Nmap ou encore de nombreux scripts qui permettent d'anticiper une attaque.

Enfin, l'utilisation excessive des fichiers de logs permet, dans de nombreux cas, de retrouver les traces d'une attaque ou de constater un dysfonctionnement. Ces derniers pourront être stockés et sur le serveur lui-même et sur un serveur de logs qui centralise ces derniers.