Sécurité SI

Authentification radius

Samedi 18 mars 2006

Germain BAUVIN (bauvin_g)
Mathieu MICHAUD (michau_m)
Pierre-Yves ROFES-VERNIS (rofes-_p)

Table des matières

Introduction

RADIUS est un protocole qui intègre les notions d'AAA (*Authorization, Authentication and Accounting*). Son but original était de permettre au FAI d'authentifier leurs clients de manière centralisée. On peut le relier à toutes sortes de services (PPP, POP, FTP, …) grâce à sa conception complètement modulaire. Les fonctionnalités d'authentification et de session couvertes par RADIUS peuvent elles mêmes être dissociées.

Ce dossier a pour but de présenter étape par étape la mise en place d'un tunnel Point à point joint à un serveur RADIUS entre deux plateformes Linux. Ainsi vous verrez les aléas inhérents à la mise en place d'une telle technologie. Cependant, le gain est réel : découpage logique et extensibilité.

Mise en Place du tunnel PPP entre les deux stations LINUX

La mise en place de l'architecture RADIUS a été faite sur 2 machines du laboratoire SRS. L'une tournant sous Linux Gentoo, l'autre sous FreeBSD. Après installation d'un démon PPTP (PoPToP) et du client associé, nous les avons configurés comme suit :

/etc/ppp/options.pptpd

```
# Verbosité
debug

# Adresse de la passerelle PPP
localip 192.168.1.254

# Plage d'adresse proposees aux clients
remoteip 192.168.1.1-253

# Offrir une configuration réseau (ici oui)
# noipparam

# DNS/WINS
ms-dns 1.2.3.4 # primaire
ms-dns 5.6.7.8 # secondaire (jusqu'à 6)
ms-wins 1.2.3.4 # idem
ms-wins 5.6.7.8

# Methodes d'authentification proposees aux clients

# BSD licensed ppp style
require-pap
refuse-chap
refuse-mschap
refuse-mschap-v2
refuse-mppe-128

# OpenSSL licensed ppp style
+pap
-chap
-chapms
-chapms-v2
-mppe-40        # soit 40-bit soit 128-bit, pas les deux
-mppe-128
-mppe-stateless

# Agir comme un proxy ARP ? (non, nous voulons du routage)
-proxyarp
```

La première difficulté rencontrée est qu'il existe différentes implémentations de PPP, et que le format du fichier de configuration diffère de l'une a l'autre. L'implémentation BSD utilise require/refuse, tandis que la seconde utilise +/-. Il faut donc connaître l'implémentation de PPP qui est installée sur le système pour pouvoir utiliser le format correspondant dans le fichier de configuration.

On active une journalisation verbeuse, afin d'obtenir un maximum d'informations dans les fichiers journaux. L'analyse de ces fichiers journaux s'est en effet révélée indispensable à l'identification et la résolution des problèmes.

Optionnellement nous pouvons ajouter :

Pour désactiver la compression des trames :

```
nobsdcomp
```

Et enfin, pour désactiver le log sur la sortie d'erreur.

```
nologfd
```

L'ouverture d'un tunnel PPTP consiste en fait en l'établissement d'une session PPP sur un socket TCP. Le démon PoPToP agit comme un point d'entre par TCP et renvoie le contenu des paquets transportés vers un serveur PPP qu'il instancie avec les options de configurations définis dans le fichier ci-dessus. PoPToP agit en fait comme une surcouche de PPP, d'où la dépendance décrite ci-dessus.

Une session PPP s'établit en trois étapes : initialisation, authentification, configuration réseau. La RFC 1661 décrit le comportement d'un serveur PPP comme un automate à nombre d'états finis dont les transitions sont déclenchées par des événements et des actions. Par exemple, l'événement « perte de lien de la sous-couche réseau » (comme lorsque l'on débranche le câble), déclenche la réinitialisation de l'automate (session fermée).

La première étape est régie par un sous-protocole de PPP, *Link Control Protocol* (LCP). Des informations de configuration sont échangées et la session est ouverte si un agrément pair à pair s'effectue. A ce stage, aucune configuration des couches réseaux ultérieurement transportées n'entre en jeu. Il s'agit uniquement d'une vérification bilatérale des capacités pour lancer l'étape suivante.

L'étape suivante, l'authentification, est facultative. Dans notre cas, avant la mise en place de RADIUS, elle était systématiquement acceptée. Si elle est nécessaire, le client soumet les informations requises selon le protocole choisi parmi ceux qui sont disponibles, c'est-à-dire ceux déclarés comme utilisables par l'étape précédente. Ce sont, entre autres, PAP, CHAP, MS-CHAP et EAP en utilisant toujours LCP. Parallèlement, un test de qualité de ligne peut-être effectué pour ajuster certains réglages des protocoles de bas niveau grâce à un autre sous protocole de PPP, Link Quality Request (LQR).

La dernière étape, la configuration réseau, s'accomplit avec encore un autre sous protocole, Network Configuration Protocol (NCP). Pour chaque protocole que le client veut encapsuler dans le tunnel, il peut demander des paramètres. Pour Ipv4, c'est avec IPCP, pour Ipv6 c'est avec IPV6CP. C 'est ainsi que le client obtient une adresse, une passerelle et des routes.

Ajout de l'authentification RADIUS

Serveur PPP

Les deux implémentations de PPP utilisées pour nos tests permettent de faire appel à un serveur d'authentification RADIUS. Par défaut, le protocole PPP n'impose pas d'authentification du correspondant. Il est possible de forcer celle-ci après l'établissement de la connexion et avant la négociation des paramètres réseau. En pratique, il s'agit de l'ajout d'une ligne dans le fichier de configuration du serveur PPP :

/etc/ppp/ppp.conf :

```
pptp:                          # label de la configuration serveur
set radius /fichier/de/configuration/client/radius
```

/fichier/de/configuration/client/radius :

```
auth rad-auth.test.com secret1     # serveur d'authentification
acct rad-acct.test.com secret2     # serveur d'accounting
```

Le serveur PPP, le NAS du point du vue de l'initiateur de la session, est un client du serveur RADIUS. Ainsi, via la méthode choisie, les coordonnées du compte sont transmises et présentées au serveur RADIUS. Comme dit précédemment, les deux fonctions d'identification + authentification et session peuvent être séparées.

Serveur d'autorisation et d'authentification

La configuration de FreeRADIUS, une implémentation libre du protocole RADIUS, a été assez complexe. Ce service est par nature très modulaire et découpé en beaucoup de fichiers de configuration. Heureusement, des exemples de chaque fichier sont fournis dans la distribution officielle et ils sont largement commentés. Le processus de configuration s'est en grande partie résumé à copier les fichiers un par un, à comprendre leur utilité et à garder les parties utiles.

Le fichier suivant classifie les clients selon leur adresse d'émission pour définir un contrôle d'accès par groupe protégé par un mot de passe.

clients.conf :

```
client 10.226.3.0/24 {
        secret =        chiche
        shortname =     barry
        nastype =       other
}
```

radiusd.conf :

```
# Affectation de variable, aide a la configuration
prefix = /usr/local
exec_prefix = ${prefix}
sysconfdir = ${prefix}/etc
localstatedir = /var
sbindir = ${exec_prefix}/sbin
logdir = /var/log
raddbdir = ${sysconfdir}/raddb
radacctdir = ${logdir}/radacct
confdir = ${raddbdir}
run_dir = ${localstatedir}/run/radiusd
log_file = ${logdir}/radius.log
libdir = ${exec_prefix}/lib
pidfile = ${run_dir}/radiusd.pid

# Le nom (ou le #id) du proprietaire du processus radiusd
user = _radius
group = _radius

# limitations de ressources
max_request_time = 30
cleanup_delay = 5
max_requests = 1024

thread pool {
        start_servers = 5
        max_servers = 32
        min_spare_servers = 3
        max_spare_servers = 10
        max_requests_per_server = 0
}

bind_address = 10.226.3.10    # ou nom, ici goudale.srs.lab.epita.fr
port = 0                      # automatique

hostname_lookups = no
allow_core_dumps = no

# Quelles informations doit-on ecrire dans le journal ?
log_stripped_names = no
log_auth = yes
log_auth_badpass = no
log_auth_goodpass = no

security {
        max_attributes = 200
        reject_delay = 1
        status_server = no
}
```

```
proxy_requests  = no    # Agir comme un proxy ?
snmp    = no            # Publier l'etat pour collecte SNMP ?

# Inclusion des options de contrôle d'acces
$INCLUDE  ${confdir}/clients.conf

# Configuration des modules
# (des qu'un module est declare, son nom peut être reutiliser)
modules {
 unix { }

  preprocess {
     huntgroups = ${confdir}/huntgroups
     hints = ${confdir}/hints
     with_ascend_hack = no
     ascend_channels_per_line = 23
     with_ntdomain_hack = no
     with_specialix_jetstream_hack = no
     with_cisco_vsa_hack = no
 }

 # Affecte des fichiers a des definitions de methodes
 files {
      usersfile = ${confdir}/users                  # identification
      preproxy_users = ${confdir}/preproxy_users    # idem proxy
      acctusersfile = ${confdir}/acct_users         # session
     compat = no
 }

 # Champs RADIUS pour detecter qu'une session est unique
 acct_unique {
      key = "User-Name,NAS-IP-Address,Client-IP-Address,NAS-Port"
 }

 # Comme utmp d'UNIX pour une session RADIUS
 radutmp {
     filename = ${logdir}/radutmp
     username = %{User-Name}
     case_sensitive = yes
     check_with_nas = yes
     perm = 0600
     callerid = "yes"
 }

  expr { }
  digest { }

  exec {
      wait = no
     input_pairs = request
 }
```

```
 exec echo {
     wait = yes
     program = "/bin/echo %{User-Name}"
     input_pairs = request
     output_pairs = reply
 }

 ippool main_pool {
     range-start = 192.168.1.1
     range-stop =  192.168.1.253
     netmask = 255.255.255.0
     session-db = ${raddbdir}/db.ippool
     ip-index = ${raddbdir}/db.ipindex
 }
}

# Contenu des phases de dialogue
instantiate {
        exec
        expr
}

authorize {
        preprocess
        auth_log
        files
}

authenticate {
        unix
}

preacct {
        preprocess
        acct_unique
}

accounting {
        detail
        radutmp
        main_pool
}

session {
        radutmp
}

post-auth {
        main_pool
        reply_log
}
```

Le fichier dictionnaire met en correspondance des noms d'attributs connus de RADIUS avec ceux donnés dans le dialogue avec ses clients qui diffèrent parfois selon les fabricants. Il n'est pas nécessaire de donner ce fichier ici, il est fourni dans la distribution officielle et doit simplement être place dans le répertoire de configuration.

L'échange avec le NAS comporte les étapes décrites dans l'ordre donné par le fichier de configuration. Nous avons essayé deux méthodes d'authentification « Unix » et « PAM ». Nous ne donnons pas la configuration avec le module PAM. Il suffirait d'ajouter la définition dans « modules » et de remplacer « Unix » par « PAM» dans l'étape « Authentication ». Ce module n'est pas recommandé par les développeurs de FreeRADIUS car les librairies sous-jacentes comporteraient des fuites de mémoire.

Analyse de l'architecture

Intérêt d'ajouter le client RADIUS

Le serveur NAS implémente déjà un système d'authentification simplifié permettant d'utiliser quelques-unes des méthodes d'authentification utilisées par RADIUS. La question de l'intérêt de l'ajout d'un serveur plus complexe comme RADIUS est donc légitime. Le but de cette installation supplémentaire est de renforcer le AAA (*Autorisation Authentification Accounting*) du NAS. Radius est en effet plus complet et plus modulaire. Permettant de gérer des bases d'utilisateurs bien plus facilement.

Analyse de l'évolutivité de la solution

Cette solution est parfaitement évolutive. De nombreuses variations peuvent être réalisées, chaque brique de l'installation pouvant être changée, tout comme chaque protocole utilisé pour leur liaison. Ces briques sont les suivantes :

- Le serveur NAS
- Le serveur RADIUS
- La base de données d'utilisateurs

Quant aux protocoles, sont modifiables :

- Le protocole d'authentification
- Le protocole de connexion

Analyse du trafic réseau

Voici le résultat de l'analyse du trafic réseau lors de la connexion d'un client PPP sur le serveur.

Comme on le voit sur la capture d'écran de Ethereal, fournie en annexe, l'ajout du serveur RADIUS ajouter une certaine quantité de traffic supplémentaire pour son travail d'authentification.

On voit bien sur l'image, la ligne en bleu prouve que la solution est opérationnelle.

Analyse des capacités d'authentification de FreeRadius

Un des buts de RADIUS est l'authentification. Pour cela, il a besoin d'une base d'utilisateurs ayant accès au réseau. FreeRADIUS offre plusieurs modules permettant l'interopérabilité avec de nombreux systèmes existants. Il s'intègre parfaitement dans des architectures avancées comme celles qui sont construites habituellement par les fournisseurs d'accès et de services. Parmi celles-ci on trouve :

- LDAP, via l'association des attributs RADIUS et LDAP dans le fichier *ldap.attr*
- ODBC, PostgreSQL, MSSQL, MySQL ou encore Oracle, en configurant les requêtes adéquates dans les fichiers correspondants (*postgresql.conf*, *mysql.conf*, …)
- OTP, en interaction avec des systèmes d'authentification forte
- Fichiers plats, par exemple /etc/passwd des systèmes Unix ou BDB
- PAM, librairie d'authentification modulaire

- EAP, famille de modules implémentant des méthodes d'authentification particulières comme LEAP, de Cisco, ou GTC.

Il existe de nombreux protocoles pour la transmission des informations nécessaires à cette tâche délicate. Nous allons passer en revue quelques-uns des principaux.

PAP

PAP, *Password Authentication Protocol,* est l'un des protocoles d'identification les plus simples.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Option								Length							
Data…															

Une fois la connexion entre le serveur et le client établie, ce dernier envoie en boucle son couple *identifiant/Mot de passe* et ce tant que la connexion n'est pas rompue ou que le serveur d'authentification n'a pas accepté l'authentification.

Le point faible de cette méthode est le fait que le mot de passe circule en clair sur le réseau, ce qui est idéal pour des tests mais quelque peu faible au point de vue sécurité pour une utilisation normale.

On lui préfère souvent des protocoles plus forts comme les suivants qui, eux, encryptent le mot de passe avant de le transmettre.

CHAP

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Code								Identifier							
Length															
Data…															

Acronyme de *Challenge-Handshake Authentication Protocol,* Chap ressemble un peu à PAP, l'aspect cryptographique en plus.

Le champ code contient l'une des quatre valeurs suivantes : Challenge, Response, Success, Failure.

L'ouverture d'une connexion a lieu de la façon suivante :

- Le client envoie un paquet 'challenge' au serveur
- Le serveur lui répond avec un autre paquet 'challenge'
- Le client envoie un paquet 'response' contenant le mot de passe crypté en se basant sur le 'challenge'
- Le serveur lui répond par un paquet 'sucess' ou 'failed', selon que l'information obtenue correspond ou pas à son propre calcul.
- A intervalles réguliers, le serveur d'authentification renvoie un 'challenge' au client, qui répond comme ci-dessus.

Son point faible est que, même si pour vérifier la légitimité d'une connexion, il compare la valeur hachée du mot de passe reçu et celle de celui stocké localement, le deuxième doit être stocké en clair sur le serveur.

MSCHAP (v1 et v2)

Mschap est une version de CHAP revisitée par Microsoft. Il répond au standard du protocole mais en y ajoutant les points suivants :

- Il n'a pas besoin de garder une version en clair du mot de passe.
- Il ajoute au serveur d'authentification des possibilités de changement de mot de passe.
- Enfin, dans son paquet 'failed', le champ data contient un message expliquant le refus de connexion

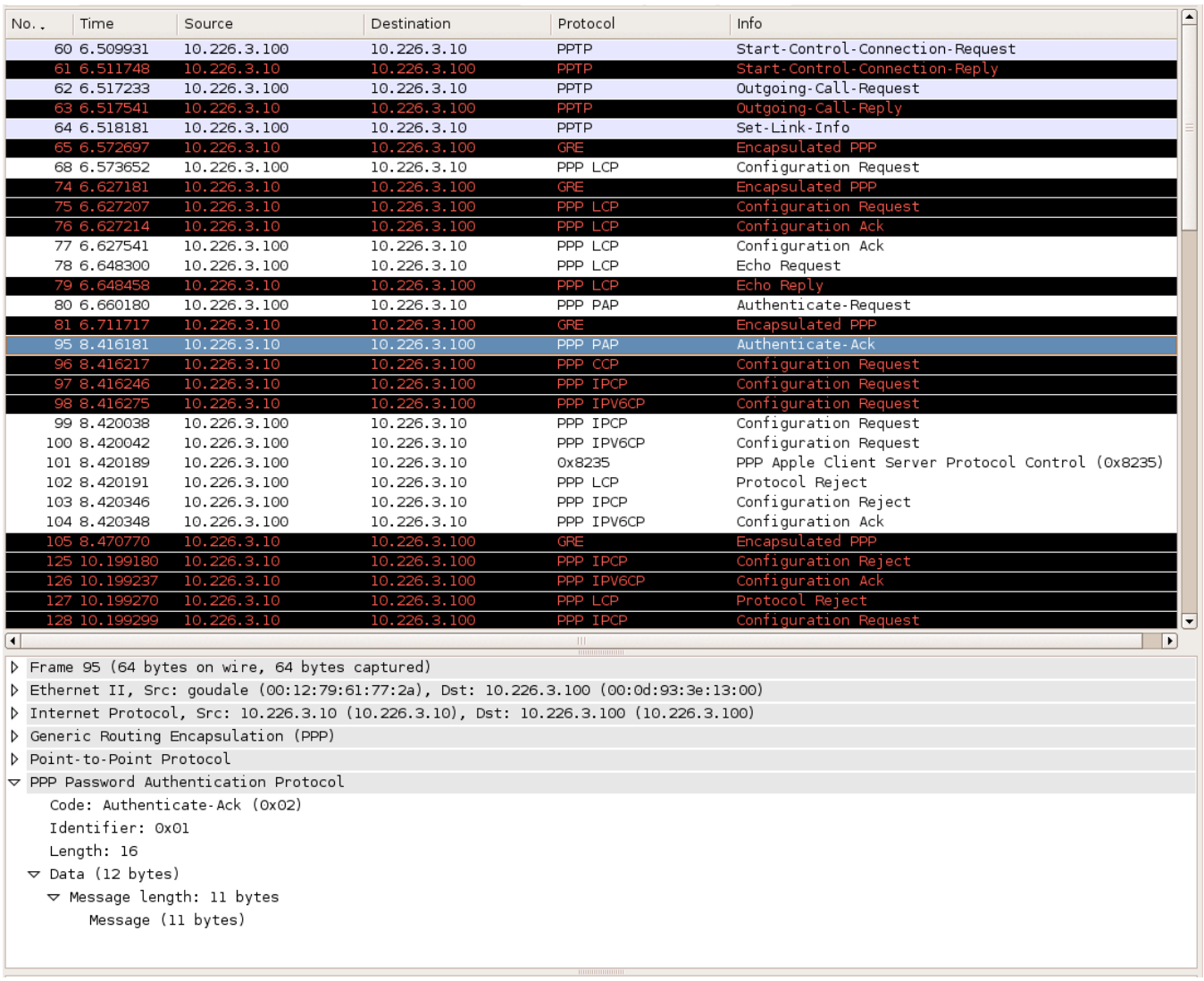
Conclusion

Malgré de nombreux problèmes de mise en place et de longues heures à scruter sans relâche les fichiers de journalisation pour les résoudre, nous avons réussi à mettre en place la solution demandée.

Ainsi nous avons pu examiner ses points forts et faibles et découvrir un peu plus l'envers de l'Internet et mettre en application des notions d'analyse réseau que nous n'avions pas forcément déjà utilisées.

Annexes

Capture ethereal



No.	Time	Source	Destination	Protocol	Info
60	6.509931	10.226.3.100	10.226.3.10	PPTP	Start-Control-Connection-Request
61	6.511748	10.226.3.10	10.226.3.100	PPTP	Start-Control-Connection-Reply
62	6.517233	10.226.3.100	10.226.3.10	PPTP	Outgoing-Call-Request
63	6.517541	10.226.3.10	10.226.3.100	PPTP	Outgoing-Call-Reply
64	6.518181	10.226.3.100	10.226.3.10	PPTP	Set-Link-Info
65	6.572697	10.226.3.10	10.226.3.100	GRE	Encapsulated PPP
68	6.573652	10.226.3.100	10.226.3.10	PPP LCP	Configuration Request
74	6.627181	10.226.3.10	10.226.3.100	GRE	Encapsulated PPP
75	6.627207	10.226.3.10	10.226.3.100	PPP LCP	Configuration Request
76	6.627214	10.226.3.10	10.226.3.100	PPP LCP	Configuration Ack
77	6.627541	10.226.3.100	10.226.3.10	PPP LCP	Configuration Ack
78	6.648300	10.226.3.100	10.226.3.10	PPP LCP	Echo Request
79	6.648458	10.226.3.10	10.226.3.100	PPP LCP	Echo Reply
80	6.660180	10.226.3.100	10.226.3.10	PPP PAP	Authenticate-Request
81	6.711717	10.226.3.10	10.226.3.100	GRE	Encapsulated PPP
95	8.416181	10.226.3.10	10.226.3.100	PPP PAP	Authenticate-Ack
96	8.416217	10.226.3.10	10.226.3.100	PPP CCP	Configuration Request
97	8.416246	10.226.3.10	10.226.3.100	PPP IPCP	Configuration Request
98	8.416275	10.226.3.10	10.226.3.100	PPP IPV6CP	Configuration Request
99	8.420038	10.226.3.100	10.226.3.10	PPP IPCP	Configuration Request
100	8.420042	10.226.3.100	10.226.3.10	PPP IPV6CP	Configuration Request
101	8.420189	10.226.3.100	10.226.3.10	0x8235	PPP Apple Client Server Protocol Control (0x8235)
102	8.420191	10.226.3.100	10.226.3.10	PPP LCP	Protocol Reject
103	8.420346	10.226.3.100	10.226.3.10	PPP IPCP	Configuration Reject
104	8.420348	10.226.3.100	10.226.3.10	PPP IPV6CP	Configuration Ack
105	8.470770	10.226.3.10	10.226.3.100	GRE	Encapsulated PPP
125	10.199180	10.226.3.10	10.226.3.100	PPP IPCP	Configuration Reject
126	10.199237	10.226.3.10	10.226.3.100	PPP IPV6CP	Configuration Ack
127	10.199270	10.226.3.10	10.226.3.100	PPP LCP	Protocol Reject
128	10.199299	10.226.3.10	10.226.3.100	PPP IPCP	Configuration Request

Liens

Une version française de la RFC :
http://abcdrfc.free.fr/rfc-vf/rfc1661.html

Le manuel PPP :
Man ppp (8)