

RFC 2865 : Remote Authentication Dial In User Service (RADIUS)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 31 Janvier 2008

Date de publication du RFC : Juin 2000

<http://www.bortzmeyer.org/2865.html>

Radius, normalisé dans ce RFC, est un protocole très simple d'AAA, conçu pour des serveurs d'accès à l'Internet (NAS, "*Network Access Server*" ou "*Access Point*" Access Point, ce dernier terme étant d'avantage utilisé dans le monde du sans-fil) qui sous-traitent l'authentification des utilisateurs à un serveur Radius. La machine qui accède à Internet n'a donc pas besoin de Radius, ce protocole ne sert qu'entre le NAS et le serveur d'authentification.

Radius fait partie de la famille des protocoles simples, certains disent simplistes, qui font peu de choses mais les font bien et, comme résultat, sont indélogeables. Normalisé d'abord dans le RFC 2058¹, puis le RFC 2138, il est désormais défini par notre RFC. Son principe est simple : lorsqu'une machine cliente se connecte au serveur d'accès (le NAS), ce dernier envoie un paquet UDP au serveur d'authentification, paquet qui contient l'information sur le client (et, dans le cas le plus simple, le mot de passe que celui-ci a indiqué). Le serveur d'accès est donc **client** Radius. Le **serveur** Radius répond par une acceptation ou un refus, accompagné, si c'est une acceptation, par la valeur de certaines options de configuration (par exemple une adresse IP).

La section 2 du RFC décrit plus en détail ce modèle. Le client Radius doit mettre dans la requête toute l'information nécessaire (nom ou NAI de l'utilisateur, numéro de port du NAS, etc). Il est également responsable, comme pour le DNS, de la réémission des demandes, s'il n'y a pas de réponse (UDP ne garantit en effet pas la délivrance des requêtes). Le RFC précise que le serveur ne doit accepter que les clients connus, configurés avec un **secret partagé**, il n'existe pas de serveur Radius public. Le serveur Radius authentifie ensuite l'utilisateur par les mécanismes de son choix (les bons serveurs Radius offrent d'innombrables techniques d'authentification).

¹Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc2058.txt>

La section 2.1 explique comment Radius peut être utilisé même si la requête ne contenait pas tous les éléments pour l'authentification, par exemple parce qu'on a choisi d'utiliser une authentification défi/réponse. Radius permet de transmettre le défi au client.

La section 2.4 est d'explication : elle donne les raisons pour lesquelles UDP a été choisi comme protocole de transport, choix qui a beaucoup été critiqué.

La section 3 est consacré au format de paquet, la 4 décrivant en détail les différents types de paquet (notamment Acceptation et Rejet). Radius utilise les ports suivants (extraits d'un `/etc/services`) :

```
radius          1812/udp
radius-acct     1813/tcp      radacct
```

C'est également cette section qui décrit le secret que partagent le client et le serveur Radius, secret qui servira notamment à xorer le mot de passe de l'utilisateur (section 5.2) pour se protéger contre l'espionnage de la ligne.

La section 5 liste tous les attributs standard, mais Radius est extensible et permet d'en décrire d'autres comme les attributs spécifiquement Microsoft du RFC 2548 ou bien comme le préfixe IPv6 délégué du RFC 4818. Cette extensibilité est une des propriétés les plus importantes de Radius.

Un des attributs les plus répandus est `User-Name` (type 1), décrit dans la section 5.1. Sa valeur est le nom de l'utilisateur (par exemple `p.dupont`) ou bien son NAI (RFC 4282, par exemple `stephane@gold.example.net`). Si c'est un nom, il est stocké en UTF-8 (les précédentes versions de Radius n'acceptaient que l'ASCII).

L'attribut `Service-Type` (section 5.6), permet de demander un service particulier, par exemple que le NAS rappelle l'utilisateur pour lui épargner les frais de communication (à Internatif, j'utilisais ainsi, pour les tâches d'administration système effectuées à distance, la valeur `Callback-Framed` à l'époque où le téléphone était presque toujours facturé à la minuté).

Les attributs sont encodés en TLV.

La section 7 du RFC donne plusieurs exemples d'échanges Radius. Prenons un cas typique :

- L'utilisateur arrive sur le NAS, qui lui demande nom (`nemo`) et mot de passe.
- Le NAS, le client Radius, demande `Access-Request`, attributs `User-Name=nemo`, `User-Password=<masqu>`, `Nas-Port=3` (pour le cas où le serveur veuille enregistrer cette information, ou même l'utiliser dans son processus de décision).
- Le serveur Radius répond `Access-Accept`, avec les attributs `Service-Type=Framed`, `Framed-Protocol=PPP` (le PPP du RFC 1661), `Framed-IP-address=255.255.255.254` (une convention courante pour dire « prends une adresse IP dynamique dans ton "pool" »), `Framed-MTU=1500`.
- Le NAS commence alors la négociation PPP avec la machine de l'utilisateur.

Notre RFC 2865 a plusieurs compagnons, notamment le RFC 2866 qui spécifie la comptabilité (le troisième A, "Accounting", du sigle AAA), et RFC 2869 le mécanisme d'extensions.

Radius a été très critiqué pour sa simplicité et un protocole concurrent, Diameter, a été développé et décrit dans le RFC 3588. Sur le créneau des serveurs d'accès à Internet, Diameter n'a eu aucun succès.

Il existe d'innombrables mises en œuvre de Radius, un protocole qu'on trouve aujourd'hui dans tous les NAS. Radius avait à l'origine été conçu par Livingston (et mes débuts personnels avec Radius, au CNAM, étaient avec un Livingston Portmaster, avant de passer, chez Internatif, à l'US Robotics Total Control) mais est aujourd'hui une norme très répandue. Livingston a depuis été rachetée par Lucent.

Côté serveur Radius, le serveur d'origine, fait par Livingston, est toujours d'actualité (paquetage `radiusd-livingston` dans Debian), mais le plus courant est désormais Free Radius, basé sur la version de Cistron. Free Radius est l'Apache des serveurs Radius : très riche en possibilités, pratiquement tout est configurable.

(Un autre serveur basé sur celui de Cistron, `xtradius` (<http://xtradius.sourceforge.net/>) ne semble plus maintenu, il est par exemple très bogué sur machines 64bits.)

Côté client (NAS), voici par exemple comment se configure un Cisco en client Radius du serveur 192.168.8.8 :

```
radius-server host 192.168.8.8 auth-port 1812 key THESHAREDSECRET aaa group ...
```

Si le NAS est une machine Unix, elle peut utiliser le greffon Radius livré avec le démon `pppd`, qui permet à `pppd` d'authentifier avec Radius.

Pour déboguer le serveur, il vaut mieux utiliser d'abord un client en ligne de commande, comme le `radclient` qu'on trouve avec Free Radius. On peut aussi facilement construire un client à soi en Python avec `pyrad` (<http://www.wiggy.net/code/pyrad/>) (le code d'exemple marche du premier coup). Restons avec `radclient` :

```
% echo "User-Name = bortzmeyer\nUser-Password = foobar" | radclient MYRADIUSSERVER auth toto
```

`tcpdump` verra alors :

```
17:31:54.679926 IP 10.1.82.1.32772 > 10.1.82.2.1812: RADIUS, Access Request (1), id: 0xcd length: 32
```

et le `radius.log` du serveur (si on veut déboguer Free Radius, il est recommandé de suivre les conseils en http://wiki.freeradius.org/index.php/FAQ#Debugging_it_yourself) :

```
Fri Jan 25 17:32:21 2008 : Error: Ignoring request from unknown client 10.1.82.1:32772
```

En effet, rappelons que Radius impose que le serveur ne communique qu'avec des clients qui ont été enregistrés. Avec Free Radius, cela se fait dans le `clients.conf` :

<http://www.bortzmeyer.org/2865.html>

```
client MYNAS.generic-nic.net {
    secret = toto
    shortname = MYNAS
}
```

Et, cette fois, ça marche :

```
% echo "User-Name = bortzmeyer\nUser-Password = foobar" | radclient MYRADIUSERVER auth toto
Received response ID 233, code 2, length = 44
    Service-Type = Framed-User
    Framed-Protocol = PPP
    Framed-IP-Address = 172.16.3.33
    Framed-IP-Netmask = 255.255.255.0
```

tcpdump a vu :

```
16:13:09.963566 IP 10.1.82.1.32774 > 10.1.82.2.1812: RADIUS, Access Request (1), id: 0xec length: 50
16:13:09.963914 IP 10.1.82.2.1812 > 10.1.82.1.32774: RADIUS, Access Accept (2), id: 0xec length: 44
```

Si on l'avait lancé avec l'option `-vvv` pour qu'il soit plus bavard :

```
16:13:25.477228 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 78) 10.1.82.1.32774 > 10.1.82.2.1812:
    Access Request (1), id: 0xee, Authenticator: eb33a21f4d8fc351898adc0b47b90c87
    Username Attribute (1), length: 12, Value: bortzmeyer
    0x0000: 626f 7274 7a6d 6579 6572
    Password Attribute (2), length: 18, Value:
    0x0000: 727b 82e4 ccab a138 a5cb 368e 8829 1555
16:13:25.477607 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 72) 10.1.82.2.1812 > 10.1.82.1.32774:
    Access Accept (2), id: 0xee, Authenticator: 779e320bca7cda5972412ee6b53b10dc
    Service Type Attribute (6), length: 6, Value: Framed
    0x0000: 0000 0002
    Framed Protocol Attribute (7), length: 6, Value: PPP
    0x0000: 0000 0001
    Framed IP Address Attribute (8), length: 6, Value: 172.16.3.33
    0x0000: ac10 0321
    Framed IP Network Attribute (9), length: 6, Value: 255.255.255.0
    0x0000: ffff ff00
```

On peut aussi tester la santé d'un serveur Radius (attention, tous ne le mettent pas en œuvre, sur Free Radius, il faut un `status_server=yes` dans la configuration) avec les requêtes de type `Status-Server` :

```
% echo "Message-Authenticator = 42" | radclient MYRADIUSERVER status toto
Received response ID 131, code 2, length = 49
    Reply-Message = "FreeRADIUS up 0 days, 00:02"
```

Si on souhaite ajouter des attributs à soi sur un serveur Radius Unix, cela se fait en général en éditant le fichier `dictionary`, qui décrit les attributs. Imaginons un attribut indiquant la température, je mets dans le `/etc/freeradius/dictionary` :

```
ATTRIBUTE      Temperature      3000      integer
```

Un exemple de réalisation d'un service Radius est décrit en <http://lehmann.free.fr/Contributions/FreeRADIUS/>