RADIUS

Conceptes (RADIUS, IEEE802.1X, EAP):

Unless a device is actually on the network, it can't use Kerberos/LDAP to authenticate user/machine accounts. But that device is already connected to the network. There's a previous, more down-level step offered by an infrastructure called RADIUS that authenticate devices at the time that they're trying to connect to a "private" network. It's most often used by ISPs to ensure only customers can connect to their dial up systems, but the chances are your wireless Access Point also supports RADIUS as a way to let authorized users connect securely (and keep unauthorized users out). It's also used on VPNs, Hotspots or even on "switchs" to give access to a wired network: the same concepts that made RADIUS work for ISPs with complex authentication requirements also make it extremely good for integrating an authentication system for devices that need to access your network.

RADIUS, which stands for “Remote Authentication Dial In User Service”, is a network protocol which serves three primary functions:

*__Authenticates__ users or devices before allowing them access to a network
*__Authorizes__/Blocks those users or devices for specific network services
*__Accounts__ for and tracks the login attempts and usage of those services

If we talk specifically about accessing to a wired network through a switch, besides a RADIUS server (running on a Linux system) we should configure that switch to be able to talk to that RADIUS server (where user authz/authn/account information is stored) but also to receive "authentication" requests from devices. This latter part is done via another protocol called IEEE 802.1X, which provisions port-based network access control over layer 2 authenticating a client when it initially connects to a LAN before it gets an IP address and additional configuration over network. So, if 802.1x is enabled on a switch port, that port will be in a blocked state until user connected to port authenticated (only 802.1x messages are allowed to go thru the port while all other packets are blocked).

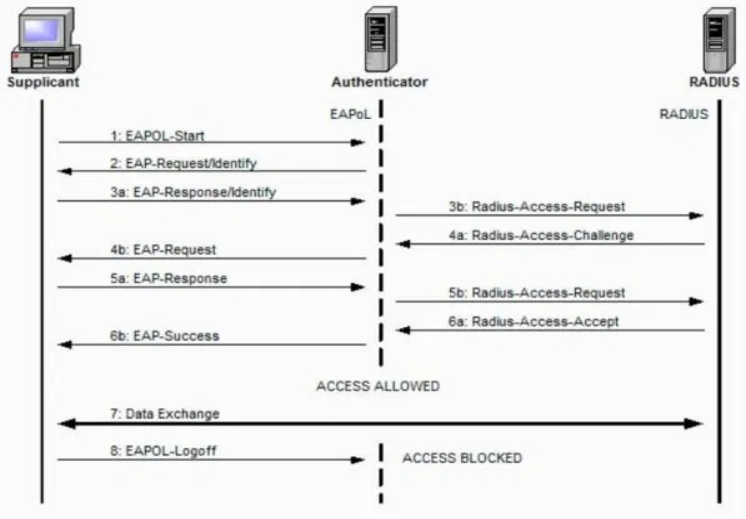
These are the major components of 802.1X:

***Supplicant:** It’s a user machine which tries to access ("to be attached to") the network. The term "supplicant" is actually used to refer to the software running on the client that provides credentials to the authenticator.
***Authentication server:** It’s a server machine which validates client’s credentials. We will use a Radius server software (there's some others, like TACACS+, but we'll only study Radius one) and EAP protocols to manage client access (by using several methods: username/passwords, certificates, ...)
***Authenticator:** It’s a L2 switch or Wireless Access Point (WAP). The job of Authenticator is to act as a proxy for supplicant (acting like a "security guard" to a protected network) and convert 802.1X message to RADIUS message and vice versa

Els protocols d'autenticació entre el supplicant i l'autenticador i entre aquest i el servidor RADIUS que tant l'estàndard 802.1X com l'estàndar RADIUS respectivament poden implementar, són diversos, però quasi tots ells estan dissenyats seguint les directrius d'una tecnologia "paraigües" anomenada EAP, la qual defineix un determinat format de missatge i un conjunt de funcions comunes que serveixen per negociar els anomenats "mètodes EAP" a usar, els quals corresponen, llavors sí, a diferents mecanismes específics d'autenticació (que determinen, entre altres coses, els diferents tipus de credencials necessaris a enviar entre el supplicant i el servidor RADIUS).

NOTA: En el cas d'autenticar dispositius inal·làmbrics, el protocol que utilitzen els AP no sol ser el 802.1X sinó el WPA2, el qual no només proporciona autenticació sinó que també aporta encriptació (de tipus simètric, mitjançant AES-CCMP) un cop l'autenticació ja s'ha fet. El WPA2 té una variant anomenada WPA2-EAP que fa que l'AP actui d'autenticador contra un servidor RADIUS de manera idèntica a la que estudiarem. D'altra banda, però, la variant de WPA2 més usada en entorns domèstics és l'anomenada WPA2-PSK, on la clau de connexió és precompartida entre el supplicant i l'AP (i no hi ha llavors servidor d'autenticació).

NOTA: Com que la tecnologia EAP va sorgir originalment per autenticar xarxes punt a punt (concretament, enllaços de tipus PPP), la seva inclusió dins dels estàndards 802.1X (i WPA2) va fer que aquesta tecnologia s'hagués d'adaptar als formats de trames de nivell 2 ja existents en les xarxes de tipus IEEE 802 (és a dir, Ethernet o WiFi). És per això que es parla de paquets EAPoL ("EAP over LAN") i no pas de paquets Ethernet o WiFi quan es vol indicar que s'està usant EAP en la xarxa IEEE 802 que hi ha entre el supplicant i l'autenticador. A partir d'aquí, l'autenticador extreurà sempre el payload del paquet EAPoL de Nivell 2 rebut i llavors l'encapsularà en forma de payload en un paquet RADIUS (de nivell 7) per enviar-lo al servidor RADIUS



Exemples de mètodes d'autenticació de tipus EAP són, entre d'altres:

*__PEAP__: This is a two stage authentication method: in first stage a secure TLS tunnel created between client and server and in second stage client is authenticated. Client authentication is done by using protocols like CHAP, PAP or -the most common- MSCHAPv2, which are user/password-based all of them. Only server side TLS-certificate is needed.

> NOTA: All the mentioned user/password-based client authentication protocols (CHAP, PAP, MSCHAPv2) transmit passwords either in clear text or hashed with some algorithm. Fortunately, PEAP (and EAP-TTLS, as we'll see in next pharagraph) encrypts them inside the TLS tunnel

*__EAP-TTLS__: EAP-TTLS is a credential-based protocol that, as in PEAP, it only requires the server to be authenticated by certificate while client authentication is optional (and it's usually in form of user/password-based protocols like PAP or MSCHAPv2, but there are multiple choices to choose). Also like in PEAP, TTLS previously creates a TLS "tunnel" between the client and the server to do the authentication.

*__EAP-TLS__: This method uses TLS handshake to mutually authenticate client and server. So, as certificate is needed on both client and server, it's great for preventing rogue/impersonation attacks. (note, however, that some tools exist that can make a certificate exportable if they are not properly protected). Moreover, it eliminates the risk of over-the-air credential theft and it also eliminates password-related disconnects due to password-change policies. Once distributed, client certificates are designed to not be removed from the device until they expire but, anyway, admins can easily revoke certificates before expiration if need be via a CRL. So, in general this method requires a full PKI infrastructure, which can be a little difficult to maintain.

> NOTA: La máxima seguridad en la conexión, la conseguimos usando certificados de cliente. Instalando un certificado digital en el equipo cliente (y evitando que se pueda exportar), nos aseguramos de que sólo los equipos con certificado se puedan conectar, eliminando las vulnerabilidades que tiene la autentificación por nombre de usuario y contraseña. El principal problema es que hay que instalar (y a ser posible supervisar la instalación para evitar que el certificado se pueda copiar a otro equipo) un certificado digital en cada uno de los equipos que vayan a usar nuestra red inalámbrica (podríamos crear un sólo certificado de cliente e instalarlo en todos los clientes que queramos, pero lo ideal es crear un certificado diferente para cada uno): esto es mucho más engorroso que simplemente añadir nombres de usuario y contraseñas a un fichero o una base de datos.

*__EAP-OTP__: Implementa contrasenyes d'un sol ús ("One-Time Passwords")

Cal tenir en compte, però, que els mètodes d'autenticació basats en contrasenya, com PAP, CHAP o MSCHAPv2, es poden implementar de forma autònoma, fora de la tecnologia EAP. No obstant, en aquest cas, la contrasenya viatjarà sense xifrar (ja sigui en texte pla -si es fa servir PAP- o bé "hasheada" amb algun algoritme com MD5 o SHA-2 -si es fa servir MSCHAPv2, per exemple-). Si es vol saber més sobre els diferents tipus d'autenticació, veieu https://support.huawei.com/enterprise/en/doc/EDOC1100086527 o https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Security/TrustSec_1-99/Dot1X_Deployment/Dot1x_Dep_Guide.html#wp386716

Posada en marxa del servidor FreeRadius

FreeRADIUS (https://freeradius.org) is an open source RADIUS suite. It ships with both server and radius client, development libraries and numerous additional RADIUS related utilities. FreeRADIUS supports request proxying, with fail-over and load balancing, as well as the ability to access many types of back-end databases. To install the server, you should only install the "freeradius" package but it's very advisable to install the "freeradius-utils" package too to have some debug/test utilities. So in Fedora you shold execute this: *sudo dnf install freeradius freeradius-utils*

> NOTA: Depending on the "backend" used by the FreeRadius server, you should install the "freeradius-mysql", "freeradius-ldap", etc packages too

> NOTA: In Debian-based systems, the server daemon is called *freeradius* instead of *radiusd* The configuration files are also located in "/etc/freeradius" folder instead of "/etc/raddb". We will use Fedora's paths/names in this guide.

> NOTA: Many router and AP OSes (like OpenWRT for instance) has a RADIUS server integrated. In this case, you could configure this one instead of a FreeRadius server on a remote Linux machine

When the server has been installed on a new machine, the first step is to generate the (required) "snake-oil" EAP-TLS certificates (more on this later):

sudo /etc/raddb/certs/bootstrap

Then, you could start the server in debugging mode:

sudo radiusd -X

This step demonstrates that the server is installed and configured properly. If the output says "Ready to process requests, then all is well". Radius server uses udp ports 1812 (for authenticate) and 1813 (for accounting). There's a complete explanation about the shown messages in https://wiki.freeradius.org/guide/radiusd-X

> NOTA: In normal circumstances, you should start Radius server as any other daemon: *sudo systemctl start radiusd*

FreeRadius suporta diferents "virtualhost" "à la Apache" gràcies a incloure'ls al seu fitxer de configuració general, "/etc/raddb/radiusd.conf" (de fet, "à la Apache" en sistemes Debian). El "virtualhost" per defecte (que és el que estem fent servir) té la configuració a l'arxiu "/etc/raddb/sites-available/default". En aquest arxiu podem trobar la secció *listen { type=auth ... }* on hi apareixen, entre altres, les directives *ipaddr=* i *port=* , que indiquen la IP i port per on escoltarà el servidor Radius peticions d'autenticació. A banda, apareix també la secció *listen { type=acct ... }* amb les mateixes directives per tal de rebre peticions d'"accounting". Els mètodes d'autenticació activats i la seva configuració particular per cadascun apareixen en un altra secció anomenada *authenticate { ... }* (en parlarem tot seguit). També apareix la secció *authorize { ... }* per gestionar eventuals filtres per autoritzacions i la secció *accounting { ... }*, entre d'altres.

Per activar un eventual "virtualhost" diferent del de per defecte simplement caldria crear un fitxer similar amb un altre nom (indicant aquest nom a la secció *server nom {...}* del principi i a continuació un accés directe del seu fitxer de configuració a la carpeta "/etc/raddb/sites-enabled/default" (a més de reiniciar el servei). Però en principi això no caldrà fer-ho perquè amb el "virtualhost" per defecte ja tindrem prou.

Definició de clients autoritzats ("authenticators")

When we discuss clients, we mean clients of the RADIUS server (such as APs, switches, routers...) NOT the network clients (such as laptops, tablets etc) as they do not talk directly to the RADIUS server. That is, we mean authenticators, not supplicants.

To do its job, the addresses of authenticators must first be allowed to connect to the RADIUS server (this should be done in RADIUS server's configuration) and, moreover, authenticators must be authenticated against RADIUS server by pointing to some simmetrical key, stored in both sides.

To add a new client to FreeRadius, you should edit the "/etc/raddb/clients.conf" file. This file basically holds the list of all the services that will allow this server to authenticate the users. You can create there a new client with any name and put in the basic configuration specifying its IP address and the secret password that is used to secure the communication between the RADIUS server and the client device. For instance:

```
client unswitchL3 {
    ipaddr = 192.0.2.1
    secret = testing789
    #virtual_server = siesvolatendreperunaltrequenosiguidefault
}
```

(where you should change the IP address to be the address of the client which will be sending Access-Request packets...it could be a network IP, like 10.0.0.0/8, too). Obviously, the client should also be configured to talk to the RADIUS server, by using the IP address of the machine running the RADIUS server, and should use the same secret as configured above in the client section.

> NOTA: The below test which will run *radtest* command from localhost will be succeful because there is a *client localhost { ... }* section in "clients.conf" file.
>
> NOTA: Inside *client { ... }* section there may be another subsection called *limit { ... }* with specific configuration limits, like this:
>
> ```
> limit {
>        max_connections = 50
>        lifetime = 0
>        idle_timeout = 30
> }
> ```

Com configurar diferents mecanismes d'autenticació a FreeRadius

The most common authentication methods are available by default in a FreeRadius server thanks to the presence of the corresponding *Auth-Type* line inside the *authenticate {...}* section in "default" virtualhost: there's one line for PAP, another for CHAP, another for MSCHAPv2, another for EAP...among others. Each of these lines load the correspoding FreeRadius module, which will have has its particular configuration located in a particular file (called "pap", "chap", "mschap" or "eap") inside the "/etc/raddb/mods-available" folder. But we must do some additional steps so that these methods work well:

*Creació d'usuaris (per fer servir PAP, CHAP, MSCHAPv2, ja sigui dins de PEAP/EAP-TTLS o fora):

La idea de tenir usuaris RADIUS és donar a les persones usuàries de la nostra xarxa (Ethernet o WiFi) un determinat usuari i contrasenya RADIUS per tal de què puguin autenticar-se en ella (i, a partir d'aquí, obtenir eventualment una IP via DHCP i començar a comunicar-se). Si per exemple tinguéssim una xarxa WiFi sostinguda per 10 punts d'accés repartits per tot el nostre edifici però no féssim servir RADIUS per autenticar-hi els clients sinó el mètode WPA2-PSK (que recordem, funciona amb una clau única), tots els usuaris d'aquesta xarxa WiFi haurien de fer servir llavors la mateixa clau per connectar els seus dispositius. ¿Què passaria llavors si decidissis canviar aquesta clau (perquè algú "extern" l'ha endevinada, per exemple)? Doncs caldria anar AP rera AP canviant la contrasenya i a més avisar a tots els usuaris que la canviessin també. D'altra banda, amb aquest sistema no podem saber a quina hora s'ha connectat cap usuari en concret perquè tots són indistinguibles. És a dir, fer servir una única clau no és un sistema massa escalable. En canvi, si cada usuari té assignada un usuari i contrasenya per autenticar-se, si aquest hagués de canviar només caldria modificar-la un sol cop al servidor RADIUS (ja estigui aquest integrat en un router o bé en un PC

extern); a més es podria controlar quan (i des d'on) es connecta cada usuari, i si aquest usuari ja no hagués de tornar a connectar-se a la xarxa, només caldria eliminar aquest usuari de la base de dades gestionada per RADIUS i prou.

Testing authentication is simple. Edit the "/etc/raddb/mods-config/files/authorize" file and add the following line of text at the top of the file, before anything else (lines are processed in order until a match):

```
pepito Cleartext-Password := "1234"
```

After restarting the server, you could run *radtest* command from another terminal window:

radtest pepito 1234 127.0.0.1 1812 testing123

This tests user "pepito" with password "1234", on the localhost server and port 1812 using the secret shared key "testing123". This secret shared key is defined by default in the *secret=* line under the *client localhost { ... }* section from "/etc/raddb/clients.conf" file. You should see the server respond with an "Access-Accept" message because it's using PAP method directly

NOTA: If you want to reject some specific user, you should write these lines at the beginning of "/etc/raddb/mods-config/files/authorize" file (keep atention to the tabulation in the second line):

```
manolito Auth-Type := Reject
        Reply-Message := "Your account has been disabled"
```

NOTA: If there's a `Fall-Through = yes` line, it means that if there's a match, next lines will we processed anyway

NOTA: There's a `DEFAULT` user, useful to be a "cath-all" one

*Creació de certificats per poder emprar EAP (creació de la CA i generació del certificat de servidor):

To implement the PEAP, EAP-TTLS or EAP-TLS methods, we need a server TLS certificate. The "snake oil" server TLS certificate created by "bootstrap.sh" script is fine to do tests but in a real environment can't be used because supplicants's network software (like wpa_supplicant or NetworManager) don't trust them by default (unless you disable some "Validate server certificate" option or similar). So "in real world" we should create a CA-signed server TLS certificate instead. One way to do this is to create a CA certificate (which should be copied to clients so that they can trust server certificates signed by that "non oficial" CA), create a CA private key (needed to efectively sign the server certificate), create our server's private key and, finally, sign our server certificate with that CA's private key (this server certificate should be copied to clients too, so that they can trust server). To create a new CA certificate/private key and our FreeRadius server certificate/private key you should do this:

1.- Edit "/etc/raddb/certs/ca.cnf" file to change, at least, these lines...:

```
default_days      = 3650
...
input_password    = supersecretandlongpassword
output_password   = supersecretandlongpassword
...
countryName            = ES
stateOrProvinceName = Barcelona
localityName           = Santako
organizationName     = PuigCastellar
emailAddress           = admin@elpuig.xeill.net
commonName          = "Freeradius CA"
```

...and create the CA certificate (with private key integrated inside, that's the "ca.pem" file) and the private key itself (that's the "ca.key" file) by running this command inside "/etc/raddb/certs" folder...:

make ca

NOTA: Els camps "input_password" i "output_password" són les passphrases que protegeixen els certificats

2.- Edit "/etc/raddb/certs/server.cnf" file to change, at least, the same lines as above with the same values except the line *commonName*, which should have another one, like "Freeradius Server Certificate", for instance (specifically, *countryName*, *stateOrProvinceName*, *localityName* and *organizationName* fields must have the same values than in above step!). Then, create server certificate (with its own private key integrated inside, that's the "server.pem" file, or without it, that's the "server.crt" file) and the server private key itself (that's the "server.key" file) by running this command inside "/etc/raddb/certs" folder...:

make server

3.- Edit "/etc/raddb/mods-available/eap" file to change this line inside *tls-config tls-common { ... }* section:

private_key_password = supersecretandlongpassword

NOTA: Dentro de la misma sección *tls-config tls-common { ... }* del fichero "/etc/raddb/mods-available/eap" deberíamos indicarle al servidor FreeRadius que use los nuevos certificados en lugar de los que vienen por defecto, de tipo "snake oil". Para ello deberíamos asegurarnos que las directivas *private_key_file*, *certificate_file* y *ca_file* tengan como valor la ruta de los ficheros "server.pem", "server.pem" y "ca.pem" creados anteriormente, respectivamente. Hay que comprobar también que dichos ficheros tengan el propietario y permisos correctos (en principio, deberían ser root:radiusd y 644, respectivamente).

*Creació de certificats per poder emprar concretament EAP-TLS (generació dels certificats de client):

You will need to edit the "/etc/raddb/certs/client.cnf" file (changing the same lines as in "ca.cnf" file and assigning them the same values except the line *commonName*, which should have another value, like "Client One Certificate", for instance, and the lines *input_password* and *output_password*). Then, create the client certificate and private key ("client.crt", "client.key" files respectively) by running this command inside "/etc/raddb/certs" folder...:

make client

NOTA: You can create a second client certificate by reediting the "client.cnf" file (be sure to use different values for the fields *emailAddress* and *commonName* : OpenSSL creates unique certificates for each client, and will complain if you try to create two different certificates which share those fields) and reexecuting the above command

NOTA: Anyway, if you rebuild the CA certificate, you'll have to rebuild all clients (and server) certificates

*Configuració del servidor FreeRadius per escollir el mètode EAP a utilitzar per defecte:

PEAP	**1.-**Edit "/etc/raddb/mods-available/eap" file to change this line inside *eap { ... }* section...: *default_eap_type=peap* ...and to change this line inside *peap { ... }* section...: *default_eap_type=mschapv2* **2.-**Copy the "ca.pem" file to the supplicant device and import them inside its own CA ring (this can be done differently depending on the client's OS). **3.-**Restart the FreeRadius server. The "inner-tunnel" virtual-host should be enabled along "default" one
EAP-TTLS	**1.-**Edit "/etc/raddb/mods-available/eap" file to change this line inside *eap { ... }* section...: *default_eap_type=ttls*

	...and to change this line inside *ttls { ... }* section...: *default_eap_type=mschapv2* **2.-**Copy the "ca.pem" file to the supplicant device and import them inside its own CA ring (this can be done differently depending on the client's OS). **3.-**Restart the FreeRadius server. The "inner-tunnel" virtual-host should be enabled along "default" one
EAP-TLS	**1.-**Edit "/etc/raddb/mods-available/eap" file to change this line inside eap *{ ... }* section...: *default_eap_type=tls* **2.-**Copy the "ca.pem", "client.crt" and "client.key" files to the supplicant device and import them inside its own CA ring (this can be done differently depending on the client's OS). **3.-**Restart the FreeRadius server.

*Testing:

While FreeRADIUS comes with a command-line tool called *radeapclient*, by far the best EAP testing tool is the *eapol_test* program from "wpa_supplicant" package (the former doesn't even support many of the authentication methods we're discussing). This program links together the same EAP peer implementation that wpa_supplicant (the WPA2 supplicant used in most Linux systems) is using and the RADIUS authentication client code from hostapd (the software most used to implement a AP in Linux systems). In other words, it integrates IEEE 802.1X Authenticator (normally, an access point) and IEEE 802.1X Supplicant (normally, a wireless client) together to generate a single program that can be used to test EAP methods without having to setup an access point and a wireless client.

To test EAP-TTLS, for instance, we need a small config file for *eapol_test*, which we'll call "eapol_test.conf" and it can look like this:

```
network={
  key_mgmt=IEEE8021X          <--This could be WPA-EAP if using WiFi. If so, a ssid="value" line is necessary too
  eap=TTLS                    <--This could be PEAP, etc
  phase2="auth=MSCHAPV2"      <--This could be PAP, etc
  identity="pepito"
  password="1234"
  ca_cert="/etc/raddb/certs/ca.pem"
}
```

Now we can execute this command: *sudo eapol_test -c eapol_test.conf -a 127.0.0.1 -p 1812 -s testing123* We should see the SUCCESS word at the end of debugging messages

To test EAP-TLS, the command will we the same but the config file will have to have this content:

```
network={
  key_mgmt=IEEE8021X
  eap=TLS
  identity="asdfqwer"    <--Pot ser qualsevol cosa
  ca_cert="/etc/raddb/certs/ca.pem"
  client_cert="/etc/raddb/certs/client.crt"
  private_key="/etc/raddb/certs/client.key"
  private_key_passwd=supersecretandlongpasswordinclient.cnffile
}
```