

Autenticación mediante PostgreSQL en Apache y ProFTPd

Paco Brufal <2:346/3.68> pbrufal@kleenux.org

Versión: 0.2r3-netsearch

Este pequeño documento explica de manera breve cómo usar los módulos de autenticación mediante base de datos PostgreSQL que vienen con los servidores Apache (web) y ProFTPd (ftp). El Sistema Operativo usado es Linux GNU/Debian Stable (aka Potato). Este documento se distribuye SIN NINGUNA GARANTIA bajo la licencia GPL (<http://www.gnu.org>). No me responsabilizo de los posibles daños que pueda causar la ejecución de los pasos descritos en este documento.

1. Introducción

Si eres administrador de un servidor web, este documento te será muy útil, ya que el trabajo rutinario de mantener al día los usuarios, grupos y contraseñas de la gente que accede a los recursos privados del servidor web puede llegar a ser muy pesado, debido a que siempre hay ficheros con contraseñas esparcidos por todo el directorio de páginas web.

En el caso de los servidores FTP, el crear un usuario que tenga acceso por FTP implica crear un usuario de sistema, y esto tiene varios inconvenientes:

- **Shell.** Si no hemos configurado el fichero `/etc/adduser.conf` para que cree los usuarios con una **shell** del tipo `/bin/false`, todos los usuarios tendrán la **shell** por defecto, que puede ser `/bin/bash` o `/bin/sh`, con lo que el usuario tendría acceso por **telnet** o **ssh** sin darnos cuenta.
- Fichero `/etc/passwd` descontrolado. Un sistema con 20 cuentas se administra de manera cómoda, pero un sistema que tiene entre 500 ó 1000 cuentas, se hace muy pesado.
- Centralización. Si tuviésemos más de un servidor FTP, independientes, el administrar las cuentas de todos ellos sería una tarea infernal. Alguno pensará que se podría mantener solo un fichero de claves de usuarios, y que todos los servidores compartiesen ese fichero mediante NFS o CODA, pero esto significa que un usuario tendría acceso a todos los servidores, cosa que posiblemente no queramos.

El uso de la autenticación mediante base de datos PostgreSQL resuelve todos los inconvenientes anteriores de una manera eficaz, cómoda y elegante, haciendo la vida del administrador mucho más feliz y productiva :D

Veamos cómo se evitan los puntos anteriores.

- En primer lugar, al crear un usuario de web o ftp, no se crean en el sistema, sino en una base de datos que se puede encontrar en otro servidor, y dicho usuario no tendrá acceso mas que a los servicios que usen la base de datos.
- En segundo lugar, la administración de una base de datos siempre es más fácil que un fichero de sistema, si se disponen de las herramientas adecuadas (léase interfaces web o X-Window para administrar PostgreSQL).
- Una base de datos la puedes crear y moldear a tu gusto, añadiendo o quitando campos a placer, y no te tienes que restringir a los campos del fichero `/etc/passwd` o `/etc/shadow`. Lo que me refiero es que tanto Apache como ProFTPd permiten hacer búsquedas selectivas dentro de la base de datos, con lo que podemos restringir las búsquedas a campos tales como servidor (el usuario pepito solo puede acceder a los servidores 1 y 3), horarios (el usuario manolito solo tiene acceso al servidor 2 de 9 de la mañana a 4 de la tarde), niveles de acceso, etc.
- Y una ventaja más es la seguridad. La base de datos podría estar en una máquina distinta a la que ofrece los servicios de web o ftp, y dedicarle una atención especial al procedimiento de securización o **hardening** del sistema (poner un firewall entre los servidores de web/ftp y la base de datos, no ofrecer servicios de telnet, ssh, rlogin, etc...).

2. Apache: módulo `mod_auth_pgsq`

En este apartado voy a explicar de manera rápida cómo se configura el Apache y la base de datos Postgres. Pondré las

ordenes que hay que ir ejecutando y un ejemplo de los ficheros de configuración. Ahí va :)

2.1 Compilación del módulo.

Antes que nada, vemos si nuestro Apache viene de serie con el módulo **mod_auth_pgsq**. Siguiendo la estructura de ficheros de una Debian , miramos en el fichero **/etc/apache/httpd.conf** si existe una línea tal que así:

```
LoadModule pgsq_auth_module /usr/lib/apache/1.3/mod_auth_pgsq.so
```

Esta línea debe estar descomentada, y el fichero debe existir. En caso que exista el fichero, pero la línea no, escríbela en la sección de los módulos y pasa al punto siguiente. Si la línea está comentada, pero el fichero no existe, entonces deberás compilarlo aparte.

La compilación de un módulo no reviste ninguna complicación. En primer lugar debemos bajarnos la última versión disponible del sitio http://www.giuseppetanzilli.it/mod_auth_pgsq/dist/ . En el momento de escribir este artículo, la última versión es la 0.9.10. Para instalar el módulo con éxito, primero debemos asegurarnos que disponemos de los siguientes paquetes (aquí los escribo con nomenclatura Debian)

- apache
- apache-dev
- postgresql
- postgresql-clients
- postgresql-dev

Para compilar el módulo en una distribución Debian, debemos retocar un poco el código fuente. Un cambio consiste en cambiar en el fichero **mod_auth_pgsq.c** la línea

```
#include <libpq-fe.h>
```

por esta otra

```
#include <postgresql/libpq-fe.h>
```

El otro cambio consiste en editar el fichero **/usr/bin/apxs** y cambiar estas 2 líneas

```
my $CFG_LD_SHLIB      = q(); # substituted via Makefile.tmpl  
my $CFG_LDFLAGS_SHLIB = q(); # substituted via Makefile.tmpl
```

por

```
my $CFG_LD_SHLIB      = q(ld); # substituted via Makefile.tmpl  
my $CFG_LDFLAGS_SHLIB = q(-G); # substituted via Makefile.tmpl
```

Seguimos los pasos de fichero **INSTALL** que describen cómo compilar el módulo (**APXS DSO Install**). Ejecutaremos el comando

```
./configure --with-apxs=/usr/bin/apxs --with-pgsq=/usr/include/postgresql
```

para generar el fichero **Makefile**. Para compilar el módulo ejecutaremos el comando

```
/usr/bin/apxs -I/usr/local/pgsq/include \  
-L/usr/local/pgsq/lib -lpq \  
-o mod_auth_pgsq.so -c mod_auth_pgsq.c auth_pgsq_shared_stub.c
```

y lo instalaremos con

```
/usr/bin/apxs -i -a -n auth_pgsq mod_auth_pgsq.so
```

Debemos verificar que en el fichero **/etc/apache/httpd.conf** existe la línea que mencionamos al inicio de este apartado:

```
LoadModule pgsq_auth_module /usr/lib/apache/1.3/mod_auth_pgsq.so
```

Tan solo queda reiniciar el Apache y esperar que todo haya salido bien ;)

2.2 Creación de la base de datos.

Ahora vamos a crear la base de datos en PostgreSQL. Siendo **root**, ejecutamos la orden

```
su - postgres
```

para crear el usuario y la bases de datos necesarias. Debemos crear un usuario, cuyo nombre será igual que el usuario que ejecuta el servidor Apache. En el caso de Debian, el usuario se llama "www-data":

```
postgres$ createuser www-data
Enter user's postgres ID -> 10000
Is user "www-data" allowed to create databases (y/n) n
Is user "www-data" a superuser? (y/n) y
WARNING: Any user who can add users can also modify the system catalog
createuser: www-data was successfully added
Shall I create a database for "www-data" (y/n) n
don't forget to create a database for www-data
```

Seguidamente, debemos crear el fichero **wwwusers.sql**, que contiene:

```
create table usuarios (
    login varchar(255),
    password varchar(255)
);
create table grupos (
    login varchar(255),
    grupo varchar(255)
);
grant all on usuarios to "www-data";
grant all on grupos to "www-data";
```

Creamos la base de datos y el esquema de las tablas:

```
postgres$ createdb wwwusers
postgres$ psql wwwusers < wwwusers.sql
create table usuarios ( login varchar(255), password varchar(255) );
CREATE
create table grupos ( login varchar(255), grupo varchar(255) );
CREATE
grant all on usuarios to "www-data";
CHANGE
grant all on grupos to "www-data";
CHANGE

EOF
postgres$
```

2.3 Configuración básica.

Ahora ya tenemos la base de datos y el usuario "www-data". El siguiente paso es definir el fichero **.htaccess** que irá en cada directorio que queramos proteger. Además, debemos asegurarnos que Apache está bien configurado para que maneje correctamente estos ficheros. En especial nos fijaremos en las directivas

```
AllowOverride none

AccessFileName .htaccess

<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

El fichero **.htaccess** contiene la siguientes líneas:

```
# máquina que contiene la base de datos
Auth_PG_host localhost
# puerto tcp de conexión
Auth_PG_port 5432
# nombre de la base de datos
Auth_PG_database wwwusers
# nombre de la tabla de usuario
Auth_PG_pwd_table usuarios
# nombre del campo de login
Auth_PG_uid_field login
```

```

# nombre del campo de password
Auth_PG_pwd_field password
# nombre de la tabla de grupos
Auth_PG_grp_table grupos
# nombre del campo de grupo
Auth_PG_gid_field grupo
# descripción de la zona protegida
AuthName "Autenticación de Usuario"
# tipo de autenticación
AuthType basic

# limitaciones del grupo administrador
<LIMIT GET POST>
require group administrador
</LIMIT>

```

Explicación rápida: en esa zona privada solo podrán acceder las personas que pertenezcan al grupo **administrador**. Es decir, aquellos logins que tengan el grupo **administrador** en la tabla grupos.

Vamos a probar el invento. Copiamos el fichero **.htaccess** anterior al raíz de nuestros documentos web (**/var/www** en Debian), y como usuario **postgres**, ejecutamos las siguientes ordenes:

```

postgres$ psql wwwusers
Welcome to the PostgreSQL interactive sql monitor:
Please read the file COPYRIGHT for copyright terms of PostgreSQL
[PostgreSQL 6.5.3 on i686-pc-linux-gnu, compiled by gcc 2.95.2]

type \? for help on slash commands
type \q to quit
type \g or terminate with semicolon to execute query
You are currently connected to the database: wwwusers

wwwusers=> insert into usuarios values ( 'admin','w8Yv0sd2meW3Q' );
INSERT 22880 1
wwwusers=> insert into grupos values ( 'admin','administrador' );
INSERT 22881 1
wwwusers=> \q

```

La contraseña **'w8Yv0sd2meW3Q'** se genera con el comando **mkpasswd**, que viene con cualquier distro.

Ha llegado el momento de reiniciar el Apache. Después de reiniciarlo con el navegador visitaremos la dirección de nuestro servidor, y nos debería aparecer la ventana de autenticación. Ponemos como login **admin**, y como password **admin**, y nos debe dejar entrar.

2.4 Configuración avanzada.

En la introducción vimos que la base de datos se puede moldear a nuestro gusto. Vamos a ver cómo restringir los accesos por horas, es decir, un usuario solo puede acceder a una zona restringida durante unas horas determinadas. Sobre la base del punto anterior, tan solo debemos añadir dos campos más a la tabla **usuarios**. Los campos serán **hora_comienzo** y **hora_fin**.

La tabla quedaría de la siguiente manera

```

create table usuarios ( login varchar(255), password varchar(255), hora_comienzo time, hora_fin time );
create table grupos ( login varchar(255), grupo varchar(255) );
grant all on usuarios to "www-data";
grant all on grupos to "www-data";

```

En el fichero **.htaccess** es donde se define el horario permitido, mediante la directiva **Auth_PG_pwd_whereclause**:

```

# máquina que contiene la base de datos
Auth_PG_host localhost
# puerto tcp de conexión
Auth_PG_port 5432
# nombre de la base de datos
Auth_PG_database wwwusers
# nombre de la tabla de usuario
Auth_PG_pwd_table usuarios
# nombre del campo de login
Auth_PG_uid_field login
# nombre del campo de password
Auth_PG_pwd_field password
# nombre de la tabla de grupos
Auth_PG_grp_table grupos

```

```

# nombre del campo de grupo
Auth_PG_gid_field grupo
# horario permitido
Auth_PG_pwd_whereclause " and hora_comienzo>=date(now()) and date(now())<=hora_fin "
# descripción de la zona protegida
AuthName "Autenticación de Usuario"
# tipo de autenticación
AuthType basic

# limitaciones del grupo administrador
<LIMIT GET POST>
require group administrador
</LIMIT>

```

Cuando creemos un usuario en esta tabla, la hora de comienzo y de fin se introducen con el formato **hh:mm:ss**.

3. ProFTPD: módulo mod_pgsql

3.1 Compilación del módulo

Realmente, no existe tal módulo en ProFTPD. Para que ProFTPD soporte consultas a bases de datos PostgreSQL, debemos recompilar el programa con la configuración deseada. Si nos bajamos el código fuente desde www.proftpd.org, debemos ejecutar los siguientes comandos

```

# CFLAGS="-O2 -Wall -I ../ -I/usr/include/mysql \
-I/usr/include/postgresql" \
./configure --prefix=/usr \
--sysconfdir=/etc --localstatedir=/var/run \
--enable-autoshadow \
--with-modules=mod_pgsql:mod_ratio:mod_quota:mod_pam:mod_readme

# make
# make install

```

Veremos ahora cómo hacerlo con paquetes Debian. En la versión inestable (aka Sid), ya existe el ProFTPD con soporte para Postgres, que se llama, **proftpd-pgsql**. Veamos cómo hacer un paquete Debian para la versión estable (aka potato). Suponiendo que nos bajamos el código fuente de ProFTPD con el comando

```
apt-get source proftpd
```

entramos en el directorio que contiene el código fuente, y en el fichero **debian/rules** editamos la línea

```
EXTRAMODS = mod_sqlpw:mod_mysql:mod_ratio:mod_quota:
```

para que ponga

```
EXTRAMODS = mod_pgsql:mod_ratio:mod_quota:
```

Lo guardamos, y como root, ejecutamos el comando

```
debian/rules binary
```

Esto nos debería construir un paquete Debian, que instalaremos con el comando

```
dpkg -i proftpd-version.deb
```

Para probarlo, insertaremos esta orden en el fichero `/etc/proftpd.conf`

```
SQLAuthoritative          off
```

si al reiniciar el demonio no da ningún error, entonces es que la compilación funcionó perfectamente. En caso contrario, repasa bien tus pasos y lee los ficheros `INSTALL` que acompañan al código fuente ;)

3.2 Creación de la base de datos

La creación de la base de datos es muy similar a la que hicimos con Apache, solo se diferencia en que esta vez el usuario que accede a la base de datos puede ser cualquier que creemos, y no el que ejecuta el servidor FTP.

La tabla que crearemos será la siguiente:

```
CREATE TABLE "usertable" (  
    "userid" text,  
    "uid" int4,  
    "gid" int4,  
    "passwd" text,  
    "shell" text,  
    "homedir" text,  
    "count" int4  
);
```

- **userid** es el campo que define el login del usuario
- **uid** es el identificador de usuario. Se puede hacer que todos los usuarios tengan el mismo **uid**, o ir asignando distintos valores.
- **gid** es el identificador de grupo al que pertenece el usuario.
- **passwd** es la clave del usuario
- **shell** es la shell que tiene el usuario. Puede ser **/bin/false**, **/bin/bash**,...
- **homedir** es el directorio donde el usuario estará cuando entre por FTP.
- **count** es una variable que se va autoincrementando según las veces que entra el usuario.

Vamos a insertar un registro en la base de datos para hacer una prueba

```
insert into usertable values ('pruebaftp',12000,100,'ZWdlLgehZOxr2','/bin/bash','/tmp',0);
```

Acabamos de crear un usuario con login **pruebaftp** y password **pruebaftp** (encriptado), cuyo **uid** es 12000 y pertenece al grupo **100**. Este usuario tiene **/bin/bash** como shell, y su **\$HOME** es el directorio **/tmp**.

3.3 Configuración básica

Teniendo como base el fichero de configuración de ProFTPD (**/etc/proftpd.conf**), añadiremos las siguientes líneas:

```
SQLAuthoritative      off  
PostgresInfo          host usuario password database  
SQLEncryptedPasswords on  
SQLUserTable          usertable  
SQLUsernameField      userid  
SQLUidField           uid  
SQLGidField           gid  
SQLPasswordField      passwd  
# SQLShellField       shell  
SQLHomedirField       homedir  
SQLLoginCountField    count
```

Explico qué significa cada línea:

- **SQLAuthoritative** es la directiva que le dice a ProFTPD que si falla la autenticación por **SQL**, que se permita otro tipo de autenticación, como podría ser **PAM** (Pluggable Authentication Modules) o **LDAP** (Lighthouse Database Access Protocol). Si esta variable está a **on** y falla la autenticación, se le deniega el paso al usuario, antes de probar con otros métodos.
- **PostgresInfo** es la directiva que le dice a ProFTPD donde está la base de datos de usuarios, y cómo tiene que conectarse a ella.
- **SQLEncryptedPasswords** activada indica que las claves de la base de datos están cifradas.
- **SQLUserTable** especifica cómo se llama la tabla que contiene los datos.
- **SQLUsernameField** indica cual es el campo que contiene el nombre de usuario.
- **SQLUidField** es el campo que contiene el **uid** del usuario.
- **SQLGidField** es el campo que contiene el **gid** del usuario.
- **SQLPasswordField** indica cómo se llama el campo de la tabla que contiene el password.
- **SQLHomedirField** es la directiva que indica a ProFTPD el campo que contiene el directorio en el cual el usuario se encontrará cuando entre al sistema.
- **SQLLoginCountField** es un contador que se incrementa cada vez que el usuario entra en el sistema.

Una vez reiniciado el demonio ProFTPD, haremos un ftp a la máquina con el usuario **pruebaftp** y password **pruebaftp** y nos debe dejar entrar.

3.4 Configuración avanzada

En este apartado haremos algo muy parecido a lo que hicimos con Apache, pero en vez de restringir por horario, ahora lo haremos por servidor. Para ello, debemos crear otra tabla donde se indica a qué servidores tiene acceso un usuario.

```
CREATE TABLE "usertable" (  
    "userid" varchar(255),  
    "uid" int4,  
    "gid" int4,  
    "passwd" varchar(255),  
    "shell" varchar(255),  
    "homedir" varchar(255),  
    "count" int4  
);  
  
CREATE TABLE "servidor" (  
    "userid" varchar(255),  
    "servidor" text  
);
```

El campo que especifica el servidor al que tiene acceso cada usuario debe ser una lista con los servidores permitidos, separados por comas y sin espacios.

Insertaremos una entrada en la tabla de accesos para hacer las pruebas necesarias:

```
insert into servidor values('pruebaftp','servidor1,servidor3');
```

Luego, en el fichero `/etc/proftpd.conf` de cada servidor debemos especificar que solo se permite la entrada a los usuarios que tengan definido ese servidor en su configuración. Esto se hace con la directiva **SQLWhereClause**. Veamos un ejemplo. Esta sería la configuración de un servidor llamado **servidor1**

```
SQLAuthoritative          off  
PostgresInfo              host usuario password database  
SQLEncryptedPasswords    on  
SQLUserTable              usertable  
SQLUsernameField          userid  
SQLUidField               uid  
SQLGidField               gid  
SQLPasswordField         passwd  
# SQLShellField           shell  
SQLHomedirField           homedir  
SQLLoginCountField        count  
SQLWhereClause " userid=(select userid from servidor where servidor like '%servidor1%') "
```

Como vemos, este servidor solo permitirá la entrada a los usuarios que tengan en el campo **servidor** el identificador del servidor1.

Si realizamos un ftp a la máquina con el mismo login y password de antes, nos debe dejar entrar. Ahora cambiaremos la línea **SQLWhereClause** de la configuración, para comprobar que se nos deniega el acceso.

```
SQLWhereClause " userid=(select userid from servidor where servidor like '%servidor2%') "
```

Haremos la misma prueba cambiando **servidor2** por **servidor3** y nos debe dejar entrar. Con esto podemos asegurar que el sistema funciona perfectamente.

4. Interfaces de administración

Para que la administración de la base de datos sea más sencilla, se debería buscar algún interfaz gráfico que nos permitiese añadir, borrar o modificar las entradas en la base de datos. Si sabes programar en PHP, no te costará hacerte tu propio interfaz web, donde podrás administrar tus usuarios de manera cómoda. Si prefieres usar algo ya hecho, puedes elegir entre **phpPgAdmin**, un interfaz web bastante completo, o **PgAccess**, un programa para X-Window que te permitirá añadir, borrar o modificar cualquier campo de la base de datos. Yo particularmente me he hecho uno en PHP, ya que puedo administrar los usuarios desde cualquier sitio.

5. Consideraciones finales

Dentro del tema de autenticación por base de datos, existen muchas más posibilidades de configuración de las que he

escrito aqui. Os animo a todos a que penseis un poco e investigueis sobre el tema :)