Configuración de PostgreSQL para soporte de conexiones seguras SSL

Francisco de Borja Lopez Rio

wu@e-shell.org

En este último artículo de la serie sobre PostgreSQL vamos a ver como configurar nuestro servidor de bases de datos para que acepte conexiones cifradas gracias a OpenSSL, con esto y los dos artículos anteriores conseguiremos tener un servidor funcional, estable y seguro.

Tabla de contenidos

1. Introducción

Como ya hemos visto en artículos anteriores, PostgreSQL soporta varios tipos de conexiones al servidor, bien locales o por red (TCP/IP). Si conectamos por red, podremos establecer conexiones encryptadas con SSL, de forma que los datos que se envian a traves de la conexión establecida por tcp/ip no circulan por esa conexión en texto plano, de forma que alguien con un sniffer tenga las cosas un poco más complicadas :O).

Ya vimos también que PostgreSQL nos da la posibilidad de encriptar los passwords en md5 para los usuarios de la base de datos, por lo que en caso de establecer conexiones sin ssl, los passwords no serían visibles, sino el hash md5. Aunque nunca está demás disponer de todos los métodos de protección posibles.

2. Configuración en 3 pasos

Para poder utilizar las conexiones ssl, hemos de seguir unos sencillos pasos:

Paso 1:

Esto lo hemos hecho en el primer artículo sobre instalación y configuración de pgsql (http://alf.e-shell.org/numero1/articulo02.php).

Paso 2:

Como hemos visto en el segundo artículo sobre configuraciones avanzadas de PostgreSQL (http://alf.e-shell.org/numero2/articulo02.php), hay un parámetro de configuración que ha de ser establecido en el arranque del servidor llamado ssl, que activado permite que el daemon escuche peticiones ssl.

```
template1=# select * from pg_settings where name='ssl';
 name | setting
------+---------
 ssl  | off
(1 row)
```

Tenemos que comprobar que este a on, lo que podemos conseguir editando el postgresql.conf y añadiendo la línea:

```
ssl = true
```

Paso 3:

Al igual que sucede cuando utilizamos servicios como pop3s, https o la función STARTTLS de sendmail, necesitamos un certificado ssl y su correspondiente llave. En el caso de PostgreSQL, los busca por defecto en el directorio data (el database cluster donde PostgreSQL guarda la información).

Al igual que con los demas certificados, lo ideal es generar la llave (.key) y el certificate request (.csr), que no es mas que el certificado sin firmar, y luego enviarlo a un CA o certification autority como thawte (www.thawte.com) para que lo firme y nos devuelva el .crt que es el certificado en si. Para este artículo vamos a generar lo que se conoce como un self-signed certificate, que es un certificado firmado por nosotros mismos y que para un entorno de pruebas sirve perfectamente.

En la página ssl-tcp (http://www.postgresql.org/docs/7.4/static/ssl-tcp.html) de la documentación de la rama 7.4 de PostgreSQL (la estable en el momento de escribir este documento) podemos ver como generar el certificado, que se hace en cinco sencillos pasos:

```
# openssl req -new -text -out server.req
# openssl rsa -in privkey.pem -out server.key
# rm privkey.pem
# openssl req -x509 -in server.req -text -key server.key -out server.crt
# chown pgsql server.key && chmod og-rwx server.key
```

Una vez seguidos estos pasos sólo tenemos que parar y relanzar el server (no vale un restart), y comprobamos que efectivamente tenga activado el parametro ssl:

```
template1=# select * from pg_settings where name='ssl';
 name | setting
------+---------
 ssl  | off
(1 row)
```

Una vez activado el parámetro de SSL, el servidor automáticamente va a requerir que las conexiones se realicen utilizando ssl para cualquier conexión a base de datos a la que le asignemos una entrada hostssl en el pg_hba.conf. Para conectar por ssl con psql (el cliente de consola de pgsql) no necesitamos nada especial, nos conectamos normalmente:

```
psql -U usuario -h host nombre_db
```

y el ya se encarga de que la conexión se establezca a través de SSL.

3. Conexión a través de túneles con OpenSSH

Además de la opción ssl del PostgreSQL, siempre podemos utilizar OpenSSH para hacer port forwarding (se puede hacer con casi cualquier servicio) de forma que establezcamos un tunnel ssh entre un puerto de nuestra maquina y un puerto del servidor de bases de datos. Por ejemplo:

```
ssh -L 5432:mi_servidor:5432 mi_usuario@mi_servidor
```

Establecería un tunnel ssh entre nuestra maquina en el puerto 5432 y el host mi_servidor también en el puerto 5432, despues de iniciar una shell en el host mi_servidor. En cuanto finalicemos esa session en la shell, se cierra el tunel, pero mientras tanto podemos acceder al servidor de bases de datos:

```
psql -U pgadmin -h localhost template1
```

Toda la información enviada al servidor va a través de este tunnel ssh, también encryptado por ssl. Esta forma, no obstante, requiere que dispongamos de una cuenta con la que iniciar una sesión de shell en el server, lo que no suele ser habitual.