LinuxFocus article number 285
http://linuxfocus.org

Het verkrijgen van toegang tot PostgreSQL door JDBC via een java SSL tunnel

Kort:

Dit artikel laat je zien hoe je JDBC toegang voor PostgresSQL op Redhat 8.0 moet opstellen, en hoe je een SSL tunnel moet creëren door gebruik te maken van Sun's Java Secured Socket Extensions, om op een veilige manier toegang tot een postgres database op afstand te verkrijgen.

door Chianglin Ng
<chglin(at)singnet.com.sg>

Over de auteur:

Ik leef in Singapore, een modern multi-raciaal land gelegen in Zuidoost Azië. Ik gebruik Linux nu ongeveer twee jaar. De distributie waarmee het voor mij allemaal begonnen is was Redhat 6.2. Vandaag gebruik ik thuis Redhat 8.0. Soms gebruik ik ook Debian Woody.

Vertaald naar het Nederlands door:
samuel Derous <cyberprophet/at/linux.be>

Inleiding

Terwijl ik over postgres en JDBC aan het leren was, stuitte ik op het probleem om op een veilige manier toegang tot een database op afstand te verkrijgen door JDBC.
Connecties via JDBC zijn niet gecodeerd en een netwerkafluisteraar kan gemakkelijk gevoelige data eruit pikken. Er zijn verschillende manieren om dit probleem tegen te gaan. De handleiding van postgres

ontsluierd dat iemand postgres met SSL-ondersteuning kan compileren door gebruik te maken van SSH doorsluizing.

In plaats van één van deze methoden te gebruiken, zou ik graag java zelf gebruiken. Met Sun's Java JDK 1.4.1 wordt de java secured socket extensions (JSSE) meegeleverd, die gebruikt kan worden om SSL connecties te creëren. De JDK voorziet ook in een gereedschap om publieke/private sleutels, digitale certifcaten en sleutelbewaarplaatsen te maken. U ziet dat het relatief gemakkelijk is om een paar proxies die op Java ingesteld zijn aan te maken en die op een veilige manier netwerkdata opnieuw kunnen uitzenden.

Het opzetten van PostgreSQL voor JDBC in Redhat 8.0

De instructies die hier verschaft worden zijn voor Redhat 8.0 bedoeld, maar de algemene principes zijn net zo goed toepasbaar op andere distro's. Je dient PostgreSQL en de bijhorende JDBC drivers te installeren als je dat nog niet hebt gedaan. Op Redhat 8.0 kun je gebruik maken van het rpm commando of van de GUI package management tool. Je dient ook Sun's JDK 1.4.1 te downloaden en te installeren. Sun's JDK wordt verschaft met enkele restricties op het gebied van encryptie wat te danken is aan de US export regelgeving. Om onbegrensde kracht van encryptie te bekomen, kun je JCE (Java Cryptographic Extensions) beleidsbestanden downloaden. Bezoek de website van Sun's Java voor meer informatie.

Ik heb JDK 1.4.1 in /opt geïnstalleerd en mijn JAVA_HOME omgeving variabele opgesteld om naar mijn JDK directory te linken. Ik heb ook mijn PATH aangepast zodat hij de directory met de JDK uitvoerbare bestanden bevat. Het volgende toont de regels die toegevoegd zijn aan mijn .bash_profile bestand.

```
JAVA_HOME = /opt/j2sdk1.4.1_01
PATH = /opt/j2sdk1.4.1_01/bin:$PATH
export JAVA_HOME PATH
```

De gelimiteerde encryptiebeleidsbestanden die met Sun JDK worden meegeleverd, heb ik ook vervangen met de ongelimiteerde van JCE. Om het mogelijk te maken dat java de JDBC drivers voor postgres zou vinden, heb ik de postgres-jdbc drivers in mijn java uitbreidingsdirectory gekopiëerd (/opt/j2sdk1.4.1_01/jre/lib/ext). In Redhat 8.0 zijn de postgres-jdbc drivers in de /usr/share/pgsql/ directory terug te vinden.

Als dit je eerste postgresql installatie betreft, zul je een nieuwe database en een nieuw postgresql gebruikersaccount aan moeten maken. SU naar root en start de postgres service. Verander daarna naar de standaard postgres administrator account.

```
su root
password:******
[root#localhost]#/etc/init.d/postgresql start
[root#localhost]# Starting postgresql service: [ OK ]
[root#localhost]# su postgres
[bash]$
```

Creëer een nieuwe postgres account en database.

[bash]$:createuser
Enter name of user to add: chianglin
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) y
CREATE USER
[bash]$createdb chianglin
CREATE DATABASE

Ik heb een postgres administrator account aangemaakt die correspondeert met mijn linux gebruikersaccount,, en een database met dezelfde naam. Wanneer je het psql gereedschap uitvoert, zal het automatisch verbinding maken met een database die correspondeert met de huidige linux gebruikersaccount. Voor meer gegevens over het beheren van databases en accounts, verwijs ik je graag door naar de handleiding van postgres. Om een wachtwoord aan te maken voor je account, kun je psql opstarten en het Alter User commando gebruiken.
Log in onder je gewoonlijke gebruikersaccount en start psql. Type het volgende.

ALTER USER chianglin WITH PASSWORD ’test1234’ ;

Om tcp/ip connecties toe te laten, dien je postgresql.conf aan te passen door de tcpip_socket optie naar ’true’ te veranderen. Bij Redhat 8 is dit bestand terug te vinden in de /var/lib/pgsql/data. Verander naar root en pas het volgende aan

tcpip_socket=true

De laatste stap bestaat erin het pg_hba.conf bestand aan te passen. Dit specificeert de hosts die zich met de postgres database kunnen verbinden. Ik heb één nieuwe hostingingang toegevoegd door een loopback adres van mijn pc te specificeren en dat het wachtwoordauthenticatie gebruikt. Je dient root te zijn om dit bestand aan te passen.

host sameuser 127.0.0.1 255.255.255.255 password

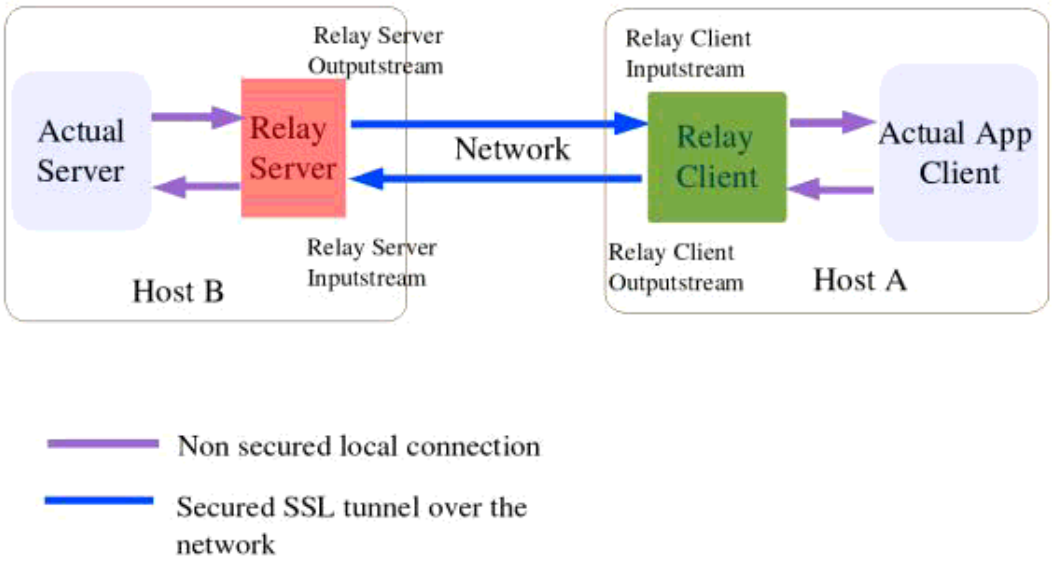
Start postgres opnieuw op en alle aanpassingen zullen verwerkt worden.

Het ontwerpen van de Java SSL Tunnel

Na deze laatste stap is postgres klaar om onbeveiligde lokale JDBC connecties te accepteren. Om op een beveiligde manier op afstand toegang te krijgen tot postgres is er een vorm van heruitzending vereist.

Het volgende diagram toont je hoe deze heruitzending zou moeten werken.

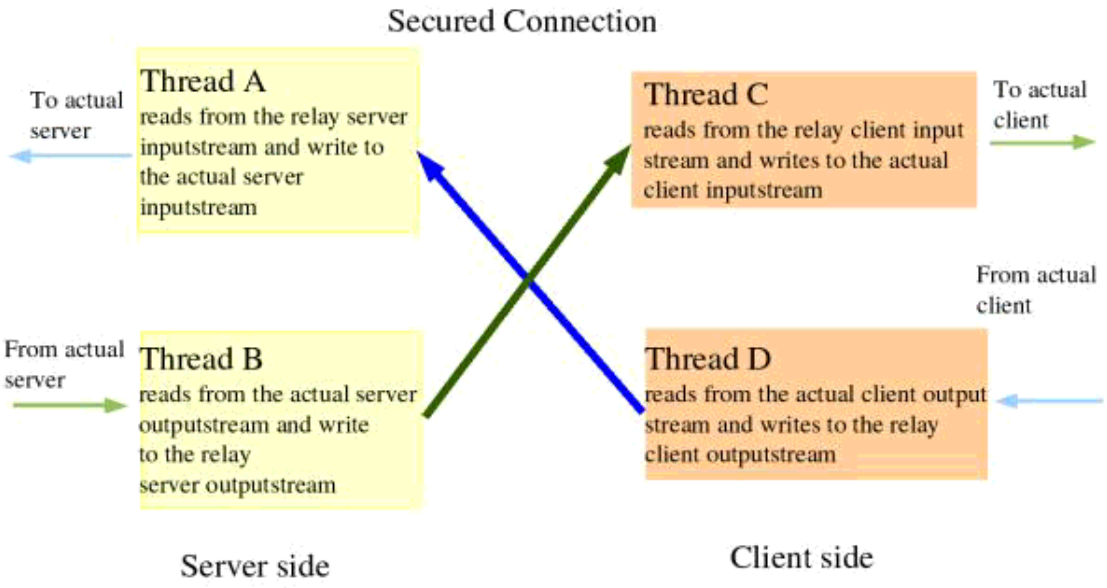
Figure 1. Client / Server Secured Relaying



De JDBC applicatie zal zich nu met de client proxy verbinden die dan al de data door een SSL connectie naar onze server proxy op afstand doorstuurd. De server proxy zal simpelweg alle pakketen naar de postgres doorzenden en antwoorden sturen via de SSL connectie terug naar de client proxy die hen dan naar de JDBC toepassing zal doorsturen. Dit volledige proces zal begrepen worden door de JDBC toepassing.

Het diagram toont ons dat aan het uiteinde van de server het een vereiste zal zijn om de data van de inkomende beveiligde stroom te krijgen en het naar de lokale uitgaande stroom te zenden die met de eigenlijke server verbonden is. Het omgekeerde is ook waar, je dient de data van de lokale inkomende stroom die verbonden is met de eigenlijke server, te verkrijgen en het de begeleiden naar de beveiligde uitgaande stroom. Ook voor de client gelden dezelfde concepten. Draden kunnen gebruikt worden om dit te implementeren. Dit toont het volgende diagram:

Fig 2. Threads needed



het ontwerpen van de sleutelbewaarplaatsen, de sleutels en de certificaten

Gewoonlijk wordt bij een SSL verbinding server-authenticatie vereist. Client-authenticatie is optioneel. In dit geval kies ik zowel voor server- als client-authenticatie. Dit betekent dat ik certificaten en sleutels zal moeten aanmaken voor zowel client als server. Ik doe dit door gebruik te maken van het sleutelgereedschap die met Java JDK wordt meegeleverd. Ik dien enkel een sleutelkast te hebben aan zowel de client als aan de server. De eerste sleutelkast is nodig om de private sleutel van de host te bewaren, en de tweede is nodig om de certificaten die door de host worden vertrouwd, in op te slaan.

De volgende lijst toont het ontwerp van een sleutelkast, een private sleutel en een publiek, zelfgetekend certificaat voor de server.

keytool -genkey -alias serverprivate -keystore servestore -keyalg rsa -keysize 2048

Enter keystore password: storepass1
What is your first and last name?
[Unknown]: ServerMachine
What is the name of your organizational unit?
[Unknown]: ServerOrg
What is the name of your organization?
[Unknown]: ServerOrg
What is the name of your City or Locality?
[Unknown]: Singapore

What is the name of your State or Province?
[Unknown]: Singapore
What is the two-letter country code for this unit?
[Unknown]: SG
Is CN=ServerMachine, OU=ServerOrg, O=ServerOrg, L=Singapore, ST=Singapore, C= [no]: yes
Enter key password for <serverprivate>
(RETURN if same as keystore password): prikeypass0 </serverprivate>

Merk op dat wachtwoorden tweemaal worden gevraagd. De eerste is voor de sleutelbewaarplaats en de tweede voor de private sleutel. Eens dit gebeurt is, exporteer dan het publieke certifcaat van de server, die door de client gebruikt zal worden om de server te authenticeren, naar een bestand.

keytool -export -alias serverprivate -keystore -rfc servestore -file server.cer

Het bovenstaande zal het zelfgetekend publiekelijk certificaat van de server in een bestand server.cer genaamd, exporteren. Aan het client-uiteinde dien je dit bestand in een sleutelkast te bewaren die alle publieke certificaten die door de client vertrouwd worden, bewaart.

keytool -import -alias trustservercert -file server.cer -keystore clienttruststore.

Het bovenstaande commando zal het publieke certificaat van de server in een sleutelkast - de clienttruststore genaamd - importeren. Als deze kast nog niet bestaat, zal die aangemaakt worden, en zal je gevraagd worden om een wachtwoord voor de bewaarplaats in te geven.

Op dit punt is je systeem in staat om een SSL verbinding te vergemakkelijken, die een server authenticatie meedraagt.
Daar ik ook de client wil authenticeren, zal ik een private/publieke sleutel voor de client dienen te maken in een nieuwe clientsleutelbewaarplaats, het client publiekelijk certificaat exporteren en deze importeren in een nieuwe sleutelbewaarplaats van de server aan het server-uiteinde.

Aan het einde van dit proces zouden er twee sleutelbewaarplaatsen moeten zijn in de server, één voor het bewaren van de geheime sleutel en een ander voor het bewaren van de certificaten die hij vertrouwt. Hetzelfde geldt voor de client.

Om de voorbeeldcode die ik later verschaf uit te voeren, is het essentiëel dat je dezelfde wachtwoorden instelt voor elke sleutelbewaarplaats dat je op een bepaalde machine ontwerpt. Dit betekent dat de twee bewaarplaatsen in de server hetzelfde wachtwoord zouden moeten hebben. Hetzelfde geldt voor de twee sleutelbewaarplaatsen in de client.

Graag verwijs ik je door naar de Documentatie van Sun om meer te leren over het gebruik van het sleutelgereedschap.

Het implementeren van de klasses

Mijn klasses zullen gebruik maken van de beveiligde socket uitbreidingen van Suns Java. De referentiegids voor Sun JSSE is beschikbaar via

http://java.sun.com/j2se/1.4.1/docs/guide/security/jsse/JSSERefGuide.html. Om een ssl verbinding te leggen dien je over een instantie van een SSLContext object te beschikken die door JSSE geleverd wordt. Initialiseer deze SSLContext met de aanpassingen die je wenst en verkrijg er een beveiligde socketFactory class van. De socketfactory kan gebruikt worden om de ssl sockets te maken.

Voor mijn implementatie zal er een client en een server proxy class zijn om de SSL tunnel aan te maken. Daar beiden een SSL verbinding zullen gebruiken zulllen ze die verwerven via een base SSLConnection class. Deze class zal verantwoordelijk zijn voor het opzetten van de initiële SSLContext die door zowel de client als de server proxies gebruikt zullen worden.

Fragment van SSLConnection klass

```
/* initKeystore om de sleutelbewaarplaatsen die de private sleutel en de vertrouwde certificaten bevat, te laden */

public void initKeyStores(String key , String trust , char[] storepass)
{
      // myKey bestaat uit mijn eigen certificaat en private sleutel, mytrust bevat alle betrouwbare certificaten.
  try {
      // Verkrijg instanties van de SUN JKS sleutelbewaarplaats
     mykey = KeyStore.getInstance("JKS" , "SUN");
     mytrust = KeyStore.getInstance("JKS", "SUN");

    // laad de sleutelbewaarplaatsen
   mykey.load(new FileInputStream(key)  ,storepass);
   mytrust.load(new FileInputStream(trust) ,storepass );
    }
 catch(Exception e) {
    System.err.println(e.getMessage());
    System.exit(1);
    }
}

/* initSSLContext methode om een SSLCcontext te verkrijgen en het met het SSL protocol en data van de sleutelbewaarplaatsen te initialiseren */
    try{
    // Verkrijg een SSLContext van Sun JSSE
    ctx = SSLContext.getInstance("TLSv1" , "SunJSSE") ;
    // initialiseer de sleutelbewaarplaatsen
    initKeyStores(key , trust , storepass) ;

    //Maak de sleutel en de trust manager factories om de certificaten af te handelen
    // in de sleutel en bewaarplaatsen
    TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509" ,
    "SunJSSE");
    tmf.init(mytrust);
```

```
    KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509" ,
    "SunJSSE");
    kmf.init(mykey , keypass);

    // Initialiseer de SSLContext met de data van de sleutelbewaarplaatsen
    ctx.init(kmf.getKeyManagers() , tmf.getTrustManagers() ,null) ;
    }
    catch(Exception e) {
    System.err.println(e.getMessage());
    System.exit(1);
    }

}
```

De initSSLContext methode creëert een SSLContext van Sun JSSE. Tijdens de creatie kun je het SSL protocol die gebruikt dient te worden specificeren. In dit geval heb ik gekozen om TLS (Transport Layer Security) versie 1 te gebruiken. Eens er een instantie van de SSLContext is verkregen, wordt het geïnitialiseerd met de data van de sleutelbewaarplaatsen.

Het volgende codeuitreksel is van de SSLRelayServer class, die op dezelfde machine zal lopen als de postgres database. Het zal alle client data die van de SSL verbinding komt doorsluizen naar postgres en vice versa.

SSLRelayServer class

```
/* De initSSLServerSocket methode zal de SSLContext via zijn super class SSLConnection verkrijgen.
Het zal dan een SSLServerSocketFactory object creëren die op zijn beurt een SSLServerSocket zal
creëren. */

public void initSSLServerSocket(int localport) {
     try{
         //Het verkrijgen van de ssl Socket factory
         SSLServerSocketFactory ssf = (getMySSLContext()).getServerSocketFactory();

          //het aanmaken van de ssl socket
         ss = ssf.createServerSocket(localport);
         ((SSLServerSocket)ss).setNeedClientAuth(true);
     }
   catch(Exception e) {
     System.err.println(e.getMessage());
     System.exit(1);
    }
 }

// beginnen met luisteren op de SSLServerSocket en wachten voor binnenkomende client connecties
public void startListen(int localport , int destport) {

    System.out.println("SSLRelay server started at " + (new Date()) + "  " +
```

```
                "listening on port " + localport + "  " +  "relaying to port " + destport );

 while(true) {
      try {
         SSLSocket incoming = (SSLSocket) ss.accept();
         incoming.setSoTimeout(10*60*1000); // bepaal een time out van 10 seconden
         System.out.println((new Date() ) + " connection from " + incoming );
         createHandlers(incoming, destport); // maak twee nieuwe draden om de binnenkomende connecties
af te handelen
       }
    catch(IOException e ) {
        System.err.println(e);
        }
    }
}
```

RelayApp class, de client proxy, is gelijkaardig als de SSLRelayServer. Het krijgt kracht van de SSLConection en gebruikt twee draden om het eigenlijke doorstsluizen te doen. Het verschil is dat het veeleer een SSLSocket creëert om met de host op afstand te verbinden dan een SSLServerSocket om naar inkomende verbindingen te luisteren. De laatste class die we nodig hebben is de draad die het werkelijke doorsluizen doet. Het schrijft simpelweg data van een inputlijn en schrijft het naar een outputlijn.

De volledige voorbeeldcode voor de vier classes is beschikbaar hier (example285-0.1.tar.gz).

Het uitvoeren en testen van de proxies

Op de client, zul je deze bestanden nodig hebben: SSLConnection.java, RelayIntoOut.java en RelayApp.java. Aan de server kant heb je SSLRelayServer.java, RelayIntoOut.java en SSLConnection.java nodig. Steek ze tezamen in een directory. Om de client proxy te compileren, gebruik je het volgende commando.

javac RelayApp.java

Om de server proxy te compileren, gebruik het volgende

javac SSLRelayServer.java

Op je server - die waarop postgres loopt - kun je SSLRelayServer doen opstarten met zes commandoregels. Ze zijn

1. Het volledige pad naar de sleutelbewaarplaats die de private sleutel van de server die je vroeger met het sleutelgereedschap hebt gemaakt, bijhoudt.
2. Het volledige pad naar de sleutelbewaarplaats van de server die het betrouwbare certificaat naar de client bijhoudt.
3. Het wachtwoord voor je sleutelbewaarplaatsen
4. Het wachtwoord voor de private sleutel van je server

5. De poort waarop deze relay server zal luisteren
6. De poort waarop data doorgestuurd kan worden (de poort van de eigenlijk server, in dit geval postgresql die als default 5432 meedraagt)

SLRelayServer servestore trustclientcert storepass1 prikeypass0 2001 5432

Eens de server proxy loopt, kun je de client proxy uitvoeren. De client proxy heeft zeven argumenten nodig, de bijkomende is de hostnaam of IP adres van de server waarop je verbinding zoekt. De argumenten zijn

1. Het volledige pad naar de sleutelbewaarplaats die de private sleutel van de client bijhoudt.
2. Het volledige pad naar de sleutelbewaarplaats van de client die het betrouwbare certificaat van de server bijhoudt
3. Het wachtwoord van je sleutelbewaarplaats
4. Het wachtwoord van de private sleutel van de client
5. De hostnaam of IP adres van de server
6. Het poortnummer van de bestemingsrelay server (in het bovenstaand voorbeeld is dit 2001)
7. Het poortnummer van de toepassing waarvoor je doorsluist, in dit geval is dit postgresql, dus zou je het moeten aanpassen naar 5432

relayApp clientstore trustservercert clistorepass1 cliprikeypass0 localhost 2001 5432

Eens je SSL tunnel voor elkaar is kan je je JDBC toepassing opstarten en op de gewoonlijke manier verbinding zoeken met postgres. Het gehele doorsluizingsingsproces zal transparant zijn voor je JDBC toepassing. Dit artikel is nu al te lang dus ik zal hier geen voorbeelden van JDBC toepassingen inlassen. De postgres handleiding en de tutorials van sun bevatten vele voorbeelden van JDBC.

Als je gedurende het testen alles op één machine wilt uitvoeren, kan dat ook. Er zijn twee manieren om dit te doen, ofwel zet je je postgres database om op een andere poort te luisteren, of je kunt het poortnummer van de RelayAPP op een andere poort instellen. Ik zal deze laatste gebruiken om een eenvoudige test te uit te voeren. Sluit eerst de RelayApp af, je zult het het Kill signaal moeten verzenden door op [ctrl] c te drukken. Je gebruikt de zelfde methode om de SSLRelayServer Proxy af te sluiten.

Start RelayApp terug op met het volgende commando. De enige verandering is het laatste poortnummer, die nu 2002 is.

java RelayApp clientstore trustservercert clistorepass1 cliprikeypass0 localhost 2001 2002

De beste toepassing om bij het testen te gebruiken zou psql zelf zijn. We zullen alle psql verkeer doorsluizen naar postgres door gebruik te maken van onze tunnnel. Type het volgende commando om psql voor het testen te starten.

psql -h localhost -p 2002

Het commando vertelt psql om verbinding te maken met localhost op poort 2002 waarop onze RelayApp is aangesloten. Nadat je je postgres wachtwoord hebt ingevuld, kun je zoals gewoonlijk SQL commando’s invoeren, en de SSL verbinding dat nu alle doorsluizen doet testen.

Een opmerking over beveiliging

Het is geen goed idee om wachtwoorden als commandoargumenten mee te geven als je een pc deelt. Dat is omdat als iemand het commando ps -auxww uitvoert meteen het volledige commando van je proces kan zien, waaronder dan ook het wachtwoord. Het is beter om je wachtwoorden in een gecodeerd formulier in een ander bestand op te slaan en je java toepassingen het vandaar uit te laten lezen. Je kunt ook Java Swing gebruiken om een dialoogvenster te maken om je naar een wachtwoord te vragen.

Besluit

Het is eenvoudig om Sun JSSE te gebruiken om een SSLTunel die door postgres gebruikt kan worden, te maken. Waarschijnlijk kan zelfs elke toepassing die een beveiligde verbinding vereist, gebruik maken van deze SSL tunnel. Er zijn zoveel verschillende manieren om encryptie aan je verbinding toe te voegen, start gewoon je favoriete linux editor en begin te coderen. Plezier ermee!

Nuttige links

- Broncode voor dit artikel
- Documentatie over PostgreSQL
- Sun JSSE Specificaties
- Sun JCA specificaties
- Tutorial over beveiliging van Java

2005-01-14, generated by lfparser_pdf version 2.51