Agent: guía de instalación y uso en CentOS

Versión: 2.5
Fecha: 19/08/2019

vintegris.com

Lista de versiones

Versión	Fecha	Autor	Cambios
1.0	12/02/2017	Sergio Sánchez	Versión inicial.
2.0	13/02/2018	Pablo González	Transcripción a LaTeX.
2.1	15/02/2018	Pablo González	Cámbio de título y nueva estructura.
2.2	19/03/2018	Jordi Casas	Actualización del procedimiento con RPM.
2.3	04/06/2018	Jordi Casas	Correcciones y notas.
2.4	12/07/2019	Domingo Jiménez Liébana	Reestructuración y simplificación del documento.
2.5	19/08/2019	Domingo Jiménez Liébana	Añadido ejemplo Java con PKCS11.

Contenido

1 Introducción

Este documento explica cómo instalar y configurar Agent en sistemas operativos CentOS. Además, se explicará la puesta en marcha de Agent como motor PKCS11 para cURL.

1.1 Requisitos de software

Los requisitos para la instalación de Agent son:

- Sistema operativo CentOS 7.
- RPM de instalación de Agent.

2 Instalación y configuración de Agent

En este apartado se describe el proceso de instalación de Agent y su posterior configuración para que funcione correctamente.

2.1 Instalación de Agent

Para realizar la instalación basta con ejecutar el siguiente comando:

```
$ sudo rpm -ivh <RPM de nebulacert-agent>
```

2.2 Configuración de Agent

Toda la configuración de Agent se define en el fichero */opt/vintegris/nebulacert-agent/etc/libpkcs11.conf*. Para el correcto funcionamiento del producto, es necesario configurar la conexión con nebulaCERT y el usuario y contraseña.

2.2.1 Definir servidor de configuración

Lo primero es configurar la conexión con nebulaCERT. Para ello, es necesario activar el servidor de configuración y definir su URL en el fichero */opt/vintegris/nebulacert-agent/etc/libpkcs11.conf*:

```
EnableConfigServer=1
ConfigServer=https://agent-ansmt01.nebulaservice.net/certclient/VinRPKCSUNIX/default
```

En el ejemplo anterior se muestran los valores por defecto que se deberían utilizar en la mayoría de las instalaciones. La URL puede cambiar en caso de disponer de un perfil de configuración concreto. También se puede realizar la configuración manualmente tal y como se explica en el apartado configuración avanzada

De esta forma, no es necesario realizar configuración adicional, ya que se carga automáticamente desde el servidor.

2.2.2 Configurar usuario y contraseña

Tanto el usuario de nebulaCERT como su contraseña se deben definir por variables de entorno o en el fichero de configuración de Agent. Si están definidos de las dos formas, tienen prioridad las credenciales definidas en el fichero de configuración.

IMPORTANTE: El usuario y la contraseña pueden ser cifrados con la herramienta de cifrado (*pwdcifrada*) que es proporcionada por separado.

Las credenciales en el fichero de configuración (*/opt/vintegris/nebulacert-agent/etc/libpkcs11.conf*), se definen con los siguientes parámetros:

```
UserID=ejemplo_user
UserCredential=ejemplo_password
```

Si alguno de los dos campos, en el fichero de configuración, se dejan en blanco, su valor se recoge de las variables de entorno “PKCS11_USR” y “PKCS11_PWD” respectivamente. Las variables de entorno se definen de la siguiente forma:

```
export PKCS11_USR="ejemplo_user"
export PKCS11_PWD="ejemplo_password"
```

NOTA: El usuario configurado en Agent, solamente puede tener configurado un Token de autenticación tipo UPLDAP en nebulaSUITE.

2.2.3 Configuración avanzada (opcional)

Para realizar la configuración manualmente, en lugar de definir el servidor de configuración habría que configurar, al menos, la conexión con nebulaCERT. Para ello, es necesario definir la URL y el secreto compartido:

```
SecureMode=1
# si no se dispone de un entorno especial, esta es la URL que se debería definir
ServerList=cert-ansmt01.nebulaservice.net
# el secreto compartido debe ser provisto
SharedSecret=<shared_secret_de_nebulacert>
```

Además, hay unos parámetros adicionales que se podrían configurar. A continuación se pueden ver todos los disponibles:

Parámetros de configuración

Parámetro	Descripción	Ejemplo
[Logging]		
LogLevel	Indica el nivel de logs. Por defecto, están deshabilitados para no utilizar el disco duro. Rango de valores posibles: [0 - 50].	0
LogFile	Fichero donde se guardarán los logs en caso de estar activados.	/opt/vintegris/nebulacert-agent/log/libpkcs11.log
[Connection]		
SecureMode	Indica si la conexión con nebulaCERT es por puerto seguro (1) o inseguro (0).	1
ServerList	URL en la que se encuentra nebulaCERT.	https://cert-ansmt01.nebulaservice.net
WaitBetweenRetry	Tiempo de espera en segundos para el próximo reintento en caso de que la comunicación con nebulaCERT falle.	3
RetryEvents	Número máximo de reintentos de comunicación con nebulaCERT en caso de que la conexión no funcione correctamente.	3
ConnectionTimeout	Máximo tiempo de espera para establecer conexión con nebulaCERT en segundos.	10
ResponseTimeout	Máximo tiempo de espera para la respuesta de nebulaCERT en segundos. En caso de tener muchos certificados, se recomienda subirlo a 30 segundos.	10
SSLVerify	Indica si se verifica el certificado utilizado para la conexión segura con nebulaCERT (1) o no (0).	1
[Credential]		

Sigue en la página siguiente

Parámetros de configuración		
SharedSecret	Es el secreto compartido con nebulaCERT para securizar la comunicación. Puede ir cifrado.	@@SharedSecret@@
UserID	Es el nombre de usuario de nebulaCERT. Puede ir cifrado.	YourUser@company.com
UserCredential	Es la contraseña del usuario de nebulaCERT. Puede ir cifrado.	YourPassword
[RemoteConfiguration]		
EnableConfigServer	Indica si se utiliza el servidor de configuración (1) o no (0). Si está activado, la configuración obtenida del servidor de configuración, prevalece sobre la definifa en este fichero.	1
ConfigServer	Es la URL, sin protocolo, del servidor de configuración.	https://agent-ansmt01.nebulaservice.net/certclient/VinRPKCSUNIX/default
[Behaviour]		
MaxCerts	Es el número máximo de certificados que se soportan. Se recomienda reducir este número en la medida de lo posible para no usar recursos innecesariamente.	100
[Developer]		
CloseSessionOnInitialize	Indica que cuando se realice una llamada a la función C_Initialize de la librería PKCS11, se borrará el ticket de sesión (1) o no (0). Por defecto, no se borra (0).	0
CloseSessionOnFinalize	Indica que cuando se realice una llamada a la función C_Finalize de la librería PKCS11, se borrará el ticket de sesión (1) o no (0). Por defecto, no se borra (0).	0
CloseSessionOnLogout	Indica que cuando se realice una llamada a la función C_Logout de la librería PKCS11, se borrará el ticket de sesión (1) o no (0). Por defecto, no se borra (0).	0

3 Integración Agent con OpenSSL y cURL (opcional)

A continuación se describen los pasos para configurar Agent como motor PKCS11 para cURL.

3.1 Instalación de dependencias

Lo primero es realizar la instalación de las dependencias.

```
$ curl -O http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo rpm -ivh epel-release-latest-7.noarch.rpm
$ sudo yum install libtool gnutls-utils openssl-devel openssl-pkcs11 -y
```

NOTA: El uso del repositorio EPEL es opcional, se podría instalar el engine de PKCS11 usando el código fuente de la librería libp11.

3.2 Instalación de cURL

Es necesario instalar cURL con soporte para SSL. Una vez descargado, éste se descomprime y se instala con las siguientes instrucciones (versión del ejemplo 7.47.1, se recomienda usar la última versión disponible soportada por la distribución de GNU/Linux):

```
$ wget http://curl.haxx.se/download/curl-7.47.1.tar.gz
$ tar -xzvf curl-*.tar.gz
$ cd curl-*
$ ./configure --with-ssl --disable-ldap && make
$ sudo make install
```

NOTA: Si no existe el comando wget se puede instalar con "sudo yum install wget".

NOTA: Para poder usar la versión de curl que se acaba de instalar podria ser necesario reiniciar la sesión de consola.

3.3 Configuración de OpenSSL

Para configurar Agent como engine PKCS11 de OpenSSL se debe editar el fichero de configuración de OpenSSL. En este ejemplo está en la ruta:

```
/etc/pki/tls/openssl.cnf
```

Se debe añadir el parámetro "openssl_conf" de la siguiente forma:

```
openssl_conf = openssl_def
```

NOTA: Este parámetro debe añadirse antes de las declaraciones de secciones.

A continuación se añade la configuración de Agent como motor PKCS11 al final del mismo fichero de configuración:

```
# PKCS11 engine config #
[openssl_def]
engines = engine_section

[engine_section]
```

```
pkcs11 = pkcs11_section

[pkcs11_section]
engine_id = pkcs11
dynamic_path = /usr/lib64/engines-1.1/libpkcs11.so
MODULE_PATH = /opt/vintegris/nebulacert-agent/lib64/libpkcs11.so

init = 0
```

3.3.1 Configuración de libp11

Se debe configurar un nuevo módulo PKCS11 en libp11. Para hacerlo, es necesario crear el siguiente archivo en el directorio de configuración de módulos de libp11. En este ejemplo está en la ruta:

```
/etc/pkcs11/modules/libpkcs11.module
```

El contenido debe ser el siguiente:

```
module: /opt/vintegris/nebulacert-agent/lib64/libpkcs11.so
managed:yes
priority:1
critical:yes
```

3.4 Ejemplo de uso con cURL

NOTA: cURL impone una limitación cuando se integra con la librería PKCS11 de Agent: el número máximo de certificados por usuario no puede ser mayor de 48.

- Para obtener la URL del token PKCS11 de debe ejecutar la siguiente instrucción:

```
p11tool --list-token
```

- Para obtener la URL de los objetos certificate y private se debe ejecutar la instrucción siguiente pasando la URL obtenida en el paso anterior:

```
p11tool --login --list-all 'token_URL'
```

- La ejecución de la operativa se realiza de la siguiente forma, donde las URLs de los objetos certificado y private se obtienen del paso anterior:

```
curl --engine pkcs11 --cert-type ENG --key-type ENG -k 'WEB_URL' --cert 'certificate_object_URL' --key
    'private_object_URL'
```

NOTA: Si sale el error "(58) ssl engine cannot load client cert with id"poner una contrabarra en los dos puntos del certificate_object_URL: 'pkcs11\:model= ...'.

A Ejemplo Java de la librería PKCS11

A continuación, se puede ver un ejemplo de una aplicación Java utilizando la librería PKCS11 de nebulaCERT.

Este ejemplo funciona tanto en Windows como en Linux. Los parámetros que espera son:

1. Ruta de la librería PKCS11. Por defecto es la de Windows de 64 bits.
2. Slot del certificado a utilizar. Si no se indica slot, cogerá el primer certificado recibido.
3. Contraseña del certificado si es que tiene.

Esta aplicación carga un certificado de nebulaCERT a través de la librería PKCS11 y realiza una firma de una cadena de texto que es mostrado por la salida estandar. En los comentarios del código, se explica con un poco más de detalle los pasos.

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.KeyStore;
import java.security.KeyStore.LoadStoreParameter;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.Security;
import java.security.Signature;
import java.security.SignatureException;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.util.Enumeration;

import javax.security.auth.login.LoginException;

public class DemoPkcs11 {

    private static final int NO_SLOT_DEFINED = -1;
    private static final String DEFAULT_PKCS11_LIBRARY_LOCATION = "C:\\Vintegris\\
        nebulaCERTagent\\x64\\VinRPKCS11.dll";

    public static void main(String[] args) throws IOException, LoginException {
        // pkcs11 library location
        String pkcs11LibraryLocation = args.length > 0 ? args[0] :
            DEFAULT_PKCS11_LIBRARY_LOCATION;
        // certificate slot number to sign
        int slot = args.length > 1 ? Integer.parseInt(args[1]) : NO_SLOT_DEFINED;
        // certificate password
        char[] certificatePassword = args.length > 2 ? args[2].toCharArray() : null;

        // load pkcs11 provider
        ByteArrayInputStream configFile = getProviderConfiguration("nebulaCERT",
            pkcs11LibraryLocation, slot);
        Provider p = new sun.security.pkcs11.SunPKCS11(configFile);
        Security.addProvider(p);
```

```
        // Load certificate to KeyStore. If SLOT not defined, will be the first one
        KeyStore keystore = getKeyStore(p, certificatePassword);
        // get private key to sign
        PrivateKey privateKey = getPrivateKey(keystore, certificatePassword);

        // perform a signature
        signTest(keystore, privateKey);

        // PKCS11 logout: this will close the nebulaCERT session if Agent parameter '
            CloseSessionOnLogout' is set to 1
        keystore = null;
        ((sun.security.pkcs11.SunPKCS11)p).logout();
    }


    /**
        * Generate configuration file to instantiate a SunPKCS11 provider
        * @param providerName The provider name
        * @param pkcs11LibraryLocation The VinRPKCS11 library location
        * @param slot The certificate slot. If value is NO_SLOT_DEFINED, will get the first
            certificate
        * @return The configuration file byte array
        */
    private static ByteArrayInputStream getProviderConfiguration(String providerName, String
        pkcs11LibraryLocation, int slot) {
        String config = "name = " + providerName + "\n" +
                  "library = " + pkcs11LibraryLocation + "\n";

        if(slot != NO_SLOT_DEFINED)
            config += "slot = " + slot;

        return new ByteArrayInputStream(config.getBytes());
    }


    /**
        * Procedure to make a signature test
        * @param keystore The KeyStore with nebulaCERT certificate
        * @param privateKey The private key to sign with
        */
    private static void signTest(KeyStore keystore, PrivateKey privateKey) {
        String signAlgorithm = "SHA256withRSA";
        byte[] textToSign = "text-to-be-signed".getBytes();
        Signature signature;

        System.out.println("Algorithm to sign" + signAlgorithm);
        System.out.println("Text to be signed: " + textToSign);
        try {
            signature = Signature.getInstance(signAlgorithm, keystore.getProvider());
            signature.initSign(privateKey);
            signature.update(textToSign);
            byte[] signedText = signature.sign();
```

```
            System.out.println("Signature done!");
            System.out.println("Signed text: " + new String(signedText));
        } catch (SignatureException | InvalidKeyException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }

    }


    /**
     * Load the nebulaCERT certificate
     * @param provider The provider to retrieve certificates
     * @param certificatePassword The certificate password. null if the certificate has no
         password.
     * @return A KeyStore with the available certificates
     * @throws IOException In case of error
     */
    private static KeyStore getKeyStore(Provider provider, char[] certificatePassword) throws
         IOException {
        KeyStore keystore;

        try {
            keystore = KeyStore.getInstance("PKCS11", provider);
            keystore.load(null, certificatePassword);
            System.out.println("nebulaCERT provider loaded successfully");
        } catch (KeyStoreException | NoSuchAlgorithmException | IOException |
             CertificateException e) {
            throw new IOException("Error: " + e.getMessage());

        }
        return keystore;
    }


    /**
     * Load the certificate private key
     * @param keystore The certificate
     * @return The private key
     */
    private static PrivateKey getPrivateKey(KeyStore keystore, char[] certificatePassword) {
        System.out.println("Getting private key...");
        Enumeration<String> aliases;

        PrivateKey privateKey = null;

        try {
            aliases = keystore.aliases();
            while (aliases.hasMoreElements()) {
                String alias = aliases.nextElement();
                System.out.println("The certificate ID is: " + alias);
                privateKey = (PrivateKey) keystore.getKey(alias, certificatePassword);
            }
        } catch (KeyStoreException | UnrecoverableKeyException | NoSuchAlgorithmException e) {
```

```
            e.printStackTrace();
        }
        if (privateKey != null) {
            System.out.println("Private key retrieved");
        }
        return privateKey;
    }
}
```

vintegris
INFORMATION SECURITY

Barcelona
Bilbao
Madrid
Valencia

+34 902 362 436
info@vintegris.com
vintegris.com

HEADQUARTERS
Av. Carrilet, 3. Edificio D, 4º B
08902 L'Hospitalet de Llobregat (Barcelona)
T +34 93 432 90 98 · F +34 93 432 93 44