The Neo4j Kerberos Add-On

v1.0

This is the documentation for the Neo4j Kerberos Add-On, authored by the Neo4j Team.

1. Introduction

Kerberos is a network authentication protocol that allows the network node to prove its identity over the network. It does so by using a Key Distribution Center (KDC) to ensure that the client identity is correct. In addition to security, Kerberos also supports single sign-on. This allows for granting users access to the database after signing in to the computer, thus providing simplicity for users.

Neo4j supports the use of Kerberos, using the *Neo4j Kerberos Add-On* described in this manual.

2. Deployment

The official Neo4j Kerberos add-on can be used to extend Neo4j with Kerberos authentication. The add-on provides authentication and should be used in conjunction with another provider such as LDAP for authorization. It is available for download at the Neo4j download site.

Compatibility

The Neo4j Kerberos add-on v1.0 is compatible with Neo4j 3.1.1 and above.

We walk you through the deployment of Kerberos add-on in the following section. It consists of several steps:

- Configure Neo4j for Kerberos
- Generate the Kerberos *keytab* file
- Configure the Kerberos add-on

For file locations, please refer to Operations Manual File Locations. All relative paths in this document are resolved against the *neo4j-home* directory.

In the examples, we are using Kerberos on Windows. We assume that LDAP is used for authorization, and that the current realm (Windows domain) is `WINDOMAIN.LOCAL`.

Configure Neo4j for Kerberos

Place the *kerberos-add-on.jar* file in the *plugins/* directory of your Neo4j installation. Edit *neo4j.conf* to enable the Kerberos add-on as authentication provider.

Example 1. Configure neo4j.conf

```
#For authentication with kerberos-add-on
dbms.security.auth_providers=ldap,plugin-Neo4j-Kerberos
dbms.security.ldap.authentication_enabled=false
dbms.security.plugin.authorization_enabled=false

#For authorization with LDAP
dbms.security.ldap.host=ad.windomain.local
dbms.security.ldap.authorization.use_system_account=true
dbms.security.ldap.authorization.system_username=CN=neo4jservice,OU=IT-Services,DC=windomain,DC=local
dbms.security.ldap.authorization.system_password=PASSWORD
dbms.security.ldap.authorization.user_search_base=CN=Users,DC=windomain,DC=local
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(SamAccountName={0}))
dbms.security.ldap.authorization.group_membership_attributes=memberOf
dbms.security.ldap.authorization.group_to_role_mapping= \
  "CN=Neo4j Read Only,CN=Users,DC=windomain,DC=local"      = reader; \
  "CN=Neo4j Read-Write,CN=Users,DC=windomain,DC=local"     = publisher; \
  "CN=Neo4j Schema Manager,CN=Users,DC=windomain,DC=local" = architect; \
  "CN=Neo4j Administrator,CN=Users,DC=windomain,DC=local"  = admin; \
  "CN=Neo4j Procedures,CN=Users,DC=windomain,DC=local"     = allowed_role
```

Generate the Kerberos *keytab* file

Create a service user for the Neo4j server in the KDC. Make sure to generate a *keytab* for the Neo4j service user and place it in the *conf* directory. In this example, we name it *neo4j.ktab*. The *keytab* will be used by Neo4j.

Configure the Kerberos add-on

If you have already been using Kerberos through the Heimdal or MIT tool `kinit`, you have a *krb5.conf* file that contains information on how to reach the KDC. If you do not have such a file, or do not wish to use it, create a new *krb5.conf* file in the *conf* directory.

Example 2. Configure krb5.conf

In this example, we are using a realm of `WINDOMAIN.LOCAL` and an IP address for the KDC and admin server of `192.168.38.2`.

Replace these with your actual realm and the actual IP or domain name of your KDC and admin server, respectively.

```
[libdefaults]
    default_realm = WINDOMAIN.LOCAL

    default_tgs_enctypes = rc4-hmac rc4-hmac-exp aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha1-96
des3-hmac-sha1
    default_tkt_enctypes = rc4-hmac rc4-hmac-exp aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha1-96
des3-hmac-sha1
    permitted_enctypes = rc4-hmac rc4-hmac-exp aes128-cts-hmac-sha1-96 aes256-cts-hmac-sha1-96 des3-
hmac-sha1

[realms]
    WINDOMAIN.LOCAL = {
        kdc = 192.168.38.2
        admin_server = 192.168.38.2
    }

[domain_realm]
    .windomain.local = WINDOMAIN.LOCAL
    windomain.local = WINDOMAIN.LOCAL
```

Some cryptographic algorithms are not natively supported by Java. When using the more advanced cryptographic algorithms (for example `ASE-256`) ensure that the Java that is running Neo4j have the `Java Cryptography Extension` extension enabled.

Now create a second file called *kerberos.conf* in the *conf* directory. The `keytab` setting refers to the *keytab* file created above. The `service.principal` setting is the principal in that keytab, and the `krb5.conf` setting refers to the *krb5.conf* file (already present from the existing Kerberos installation, or created in the step above).

Again, replace the values with the correct values according to your setup.

Example 3. Configure kerberos.conf

```
realm=WINDOMAIN.LOCAL
keytab=conf/neo4j.ktab
service.principal=neo4j/neo4j.windomain.local@WINDOMAIN.LOCAL
krb5.conf=conf/krb5.conf
```

3. Usage

Client application code that is to use Kerberos for authentication must to authenticate against the KDC and acquire a service ticket for the Neo4j service (in our example: `neo4j/neo4j.windomain.local@WINDOMAIN.LOCAL`). The service ticket must use the `Kerberos v5 (1.2.840.113554.1.2.2)` mechanism either directly or wrapped in `SPNEGO (1.3.6.1.5.5.2)`.

The service ticket should be provided to the Neo4j driver in an auth token with the following properties:

- Principal: empty
- Credentials: the Base64-encoded service ticket
- Realm: `add-on-Neo4j-Kerberos`

Please note that the Kerberos add-on does not currently work with the Neo4j Browser. It only work with applications using the Neo4j drivers.

Example code

Example 4. Example using Java

This is an example implementation using the 1.3 Java driver.

```
public void connect()
{
    byte[] serviceTicket = get( serviceDomainName );

    String scheme = "ticket";
    String realm = "add-on-Neo4j-Kerberos";
    String encodedServiceTicket = Base64.getEncoder().encodeToString( serviceTicket );
  AuthToken token = AuthTokens.custom( encodedServiceTicket );

    try ( Driver driver = GraphDatabase.driver( "bolt://" + serviceDomainName, token ) )
    {
        // do interesting things
    }
}

public byte[] get( String serviceDomainName ) throws LoginException, GSSException
{
    Map<String,String> options = Collections.singletonMap( "useTicketCache", "true" );
    Krb5Configuration loginContextConfiguration = new Krb5Configuration( options );
    LoginContext loginContext = new LoginContext(
        "KerberosClient",
        null, // this is the subject
        null, // no need for this
        loginContextConfiguration
      );
    loginContext.login();

    return getServiceTicket( loginContext.getSubject(), "neo4j@" + serviceDomainName );
}
public static final Oid SPNEGO_OID = getOid( "1.3.6.1.5.5.2" );
public byte[] getServiceTicket( Subject subject, String servicePrincipalName ) throws GSSException
{
    GSSManager manager = GSSManager.getInstance();
    GSSName serverName = manager.createName( servicePrincipalName, GSSName.NT_HOSTBASED_SERVICE );
    final GSSContext context = manager.createContext(
                    serverName, SPNEGO_OID, null, GSSContext.DEFAULT_LIFETIME );
    // The GSS context initiation has to be performed as a privileged action.
    return Subject.doAs( subject, new PrivilegedAction<byte[]>()
    {
        public byte[] run()
        {
            try
            {
                // This is a one pass context initialisation.
                context.requestMutualAuth( false );
                context.requestCredDeleg( false );
                return context.initSecContext( new byte[0], 0, 0 );
            }
            catch ( GSSException e )
            {
                e.printStackTrace();
                return null;
            }
        }
    } );
}

private class Krb5Configuration extends Configuration
{
    private final AppConfigurationEntry[] configList;

    public Krb5Configuration( Map<String,String> options )
    {
        this.configList = new AppConfigurationEntry[1];
        configList[0] =
            new AppConfigurationEntry(
                "com.sun.security.auth.module.Krb5LoginModule",
                AppConfigurationEntry.LoginModuleControlFlag.REQUIRED,
                options
            );
    }

    @Override
    public AppConfigurationEntry[] getAppConfigurationEntry( String name )
    {
        return configList;
    }
}
```

Example 5. Example using C#

This is an example implementation using C# with the 1.3 .NET driver.

```
var token = AuthTokens.kerberos(getTicket("neo4j"));
    using (var driver = GraphDatabase.Driver("bolt://neo4j.windomain.local:7687", token))
    {
        try
        {
            using (var session = driver.Session())
                {
                    var result = session.Run("MATCH () RETURN count(*) AS count");
                    foreach (var record in result)
                    {
                        Console.WriteLine($"Nodecount: {record["count"].As<string>()}");
                    }
                }
        }
        catch (Exception e)
        {
            Console.WriteLine($"Error: {e.Message}");
        }
}

 private static String getTicket(string serviceName)
 {
    AppDomain.CurrentDomain.SetPrincipalPolicy(System.Security.Principal.PrincipalPolicy
.WindowsPrincipal);
    var domain = Domain.GetCurrentDomain().ToString();

    using (var domainContext = new PrincipalContext(ContextType.Domain, domain))
    {
        string spn = UserPrincipal.FindByIdentity(domainContext, IdentityType.SamAccountName,
serviceName).UserPrincipalName;
        Console.WriteLine("Service Principale name: " + spn);
        KerberosSecurityTokenProvider tokenProvider = new KerberosSecurityTokenProvider(spn);
        KerberosRequestorSecurityToken securityToken = tokenProvider.GetToken(TimeSpan.FromMinutes(
1)) as KerberosRequestorSecurityToken;
        var token = securityToken.GetRequest();
        String ticket = Convert.ToBase64String(token);
        return ticket;
    }
 }
```