Security in Neo4j

Table of Contents

About this module

Now that you have gained experience managing a Neo4j instance and database , as well as managing Neo4j Causal Clusters, you will be introduced to how to secure a Neo4j application.

At the end of this module, you should be able to:

- Describe what security means for an application.

- Configure a Neo4j instance to use LDAP as the authentication provider.

- Describe how to secure data-in-transit.

- Describe how to secure data-at-rest.

- Configure and use security auditing.

What is application security?

Many applications today are distributed and provide services and data over the Internet. Users include customers, partners, and off-site employees. In this interconnected world, security is a major concern. There are malicious actors who will attempt to access your application's data, causing disruption and financial loss to your business. There are numerous ways that can be used to attack the data via the Internet, intranet, and even the 'human' back door. Fixing a security breach may require significant updates to your application, especially if the application was not written or implemented to be secure. Deploying a newer version of your application means additional testing and possible down time, both costly to the enterprise.

In addition, your business may need to meet regulatory security requirements or security requirements promised in a customer contract. If it is discovered that your application does not meet these requirements, you could face fines or other serious consequences.

To secure your application, you need to be concerned about 3 things:

- Data security
- Access control
- User privacy

Data security

Some information in an application is private must be protected. Exposing a customer's private information could ruin the reputation of your enterprise, which could take years to recover from. Here are some examples of the types of private data your application needs to secure:

- Company financial data such as salaries, revenue, operating expenses, etc.
- User pass-phrases for the application
- Personal data such as names, credit card numbers, telephone numbers, social security numbers, etc.
- Intellectual property such as code

Access control

Controlling who accesses your application and what they have access to is very important to the business. You want to ensure that only valid users can use the application and malicious actors are kept out. In addition, you want to control which users can access specific application functionality and monitor what users do.

User privacy

Another concern in applications that extend to the Internet is user privacy. When a Web or mobile app requests data from your application, you need to ensure that only the required data is retrieved and sent to the app. That is, if the app requests personal information, such as a driver's license ID, the data returned to the app contains only the ID for the user requested and not the IDs of multiple users that the app, then has to filter for display.

Securing a Neo4j application

Neo4j applications consist of many parts, including databases, static files like images and document scans, application code, application servers and Web servers. These all need to be secured and you use different techniques and technologies to secure them.

There are many things you can do to make your Neo4j application secure. At the highest level, there are 2 areas you must address:

1. Building security into your Neo4j application.

2. Securing the deployed application and infrastructure.

Building security into your Neo4j application

The primary facets of building security into your Neo4j application include:

- Authentication — Is the user who they say they are?

- Authorization — Is a user allowed to do what they are attempting to do?

- Auditing – Create a record of who did what and when (so that you can monitor activity and investigate security breaches).

At a high level, the implementation of how your application performs authentication, authorization, and auditing can be configured by you as an administrator. However, there are aspects of security that may cross into application code. For example, there may be a specific procedure written that can only be executed by certain users. The procedure must be annotated and configured as such. In addition, code may be written to check roles at runtime for authorization.

Securing the deployed application and infrastructure

Securing a deployed Neo4j application and its infrastructure includes securing Neo4j instances and non-Neo4j server processes they communicate with, filesystems, networks, etc. This can include:

- Data-in-transit — securing data transmitted over the network.

- Data-at-rest — securing private data in Neo4j database.

- OS level resources — securing networks and filesystems.

- Server processes — Neo4j instances, application servers, connectors, and Web servers.

- Application-related files—securing application-related files outside of the database.

In this module, you will be introduced to how you as an administrator can secure those parts of the application related to Neo4j. This module will not cover tasks related to securing non-Neo4j resources, networks, and filesystems.

Neo4j software updates

As an administrator, you must be aware of all software on your production systems and keep up-to-date with the software. In particular, you <u>must</u> ensure that the Neo4j version you are using has the the latest versions and patches, especially those that address security.

Authentication in Neo4j

There are three types of authentication frameworks supported by Neo4j:

- Native user authentication

- Custom-built authentication

- Single Point of Authentication (SPA)

Native user authentication means that users are created in the Neo4j database and authentication is performed based upon those values. Most enterprise applications do not use native user authentication in their deployed application.

Your application developers could write a custom authentication plugin. Although this is possible, the underlying internal procedures called by the custom authentication plugin could change in future releases of Neo4j so it is best to avoid this type of authentication for a deployed, secure application that will survive upgrades of Neo4j.

A SPA is highly recommended by Neo4j because it is easier to maintain and is more secure than an enterprise that uses multiple sources of user accounts. Examples of a SPA are Lightweight Directory Access Protocol (LDAP), Active Directory (AD), and Kerberos which can be used for single sign-on. The SPA is the only service in your enterprise that stores user names and passwords. For training purposes, we will use an LDAP provider for authentication, but in your real application, you will need to configure Neo4j for whatever provider your application uses for user authentication.

You will use OpenLDAP for the hands-on Exercises of this module.

If you will be using a different LDAP or authentication provider in your real application, you should consult the Neo4j Operations Manual.

Enabling authentication

A secure Neo4j instance should always have authentication enabled. By default, authentication is enabled for a Neo4j instance. You can explicitly set it in **neo4j.conf** as follows:

```
dbms.security.auth_enabled=true
```

How roles are used for users

Even though roles are not required for authentication, they are for authorization to access Neo4j. Roles must be built into the configuration of a SPA, for example LDAP. Roles are used at runtime to authorize how the current user can access Neo4j resources (data or procedures).

Refer to this table in the Neo4j Operations Manual that describes Neo4j native roles. One of these pre-defined, native roles must be associated with any user that will be connecting to the Neo4j instance for general access. So for example, the user with the role of *reader* will only have read access to the database while a user with the role of *publisher* will be able to create data in the database.

In addition, you can define custom roles that are application-specific and used together with specific custom procedures.

Using LDAP for authentication and authorization

There are many LDAP providers an enterprise can use including Active Directory and Open LDAP.

The tasks for using LDAP for authentication with Neo4j include:

1. Add role information to LDAP (groups).

2. Configure Neo4j for LDAP authentication.

3. Test.

Adding groups for roles in LDAP

In your enterprise, you will need to work with the system administrator responsible for maintaining the user directory (LDAP). Part of the configuration that <u>must</u> be modified for LDAP is the addition of groups for users. The groups added to LDAP and associated with specific users correspond to roles in Neo4j. An easy way to identify a group in LDAP is with a group ID.

For example, the LDAP would have these entries that correspond to the Neo4j native *reader* and *publisher* roles for the training **neo4jtraining.com** domain. The attribute that is used to identify the groups is *gidnumber*.

```
# reader group
dn: cn=reader,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: reader
gidnumber: 501

# publisher group
dn: cn=publisher,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: publisher
gidnumber: 502
```

Associating users with groups in LDAP

In addition, each user that will be authenticated for connecting to the Neo4j instance, will need to be modified in the LDAP to be associated with a specific group. Here is are examples of the *reader* and *publisher* users that you will be working with in this training:

```
# user reader
objectClass: organizationalPerson
objectClass: person
objectClass: extensibleObject
objectClass: uidObject
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: Reader User
givenName: Reader
sn: reader
uid: reader
gidnumber: 501
uidNumber: 1000
homedirectory: /home/users/reader
mail: reader@neo4jtraining.com
ou: users
userpassword: reader

# user publisher
dn: uid=publisher,ou=users,dc=neo4jtraining,dc=com
objectClass: organizationalPerson
objectClass: person
objectClass: extensibleObject
objectClass: uidObject
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: Publisher User
givenName: Publisher
sn: publisher
uid: publisher
gidnumber: 502
uidNumber: 1001
homedirectory: /home/users/publisher
mail: publisher@neo4jtraining.com
ou: users
userpassword: publisher
```

There are many ways to configure groups and users in LDAP. This is just an example of how the LDAP is configured that you will be working with in this training. Notice that each group has an ID, *gidnumber* and each user has an ID, *uidNumber*. The group ID is used to map to roles in Neo4j.

Note that if a user does not specify a group ID, it must specify an attribute *memberOf* that is set to the designated name, *dn* of the group.

Querying the LDAP Server

The Neo4j instance never writes to the LDAP Server. Before you begin configuring the Neo4j instance to use LDAP, you should confirm that the LDAP Server is properly configured for use with Neo4j. You query the LDAP Server by using the `ldapsearch` utility from the ldap-utils library that is available on OS X and Linux, and by `ldp.exe` on Windows.

To understand the groups that users are associated with in the LDAP, you should perform this type of query:
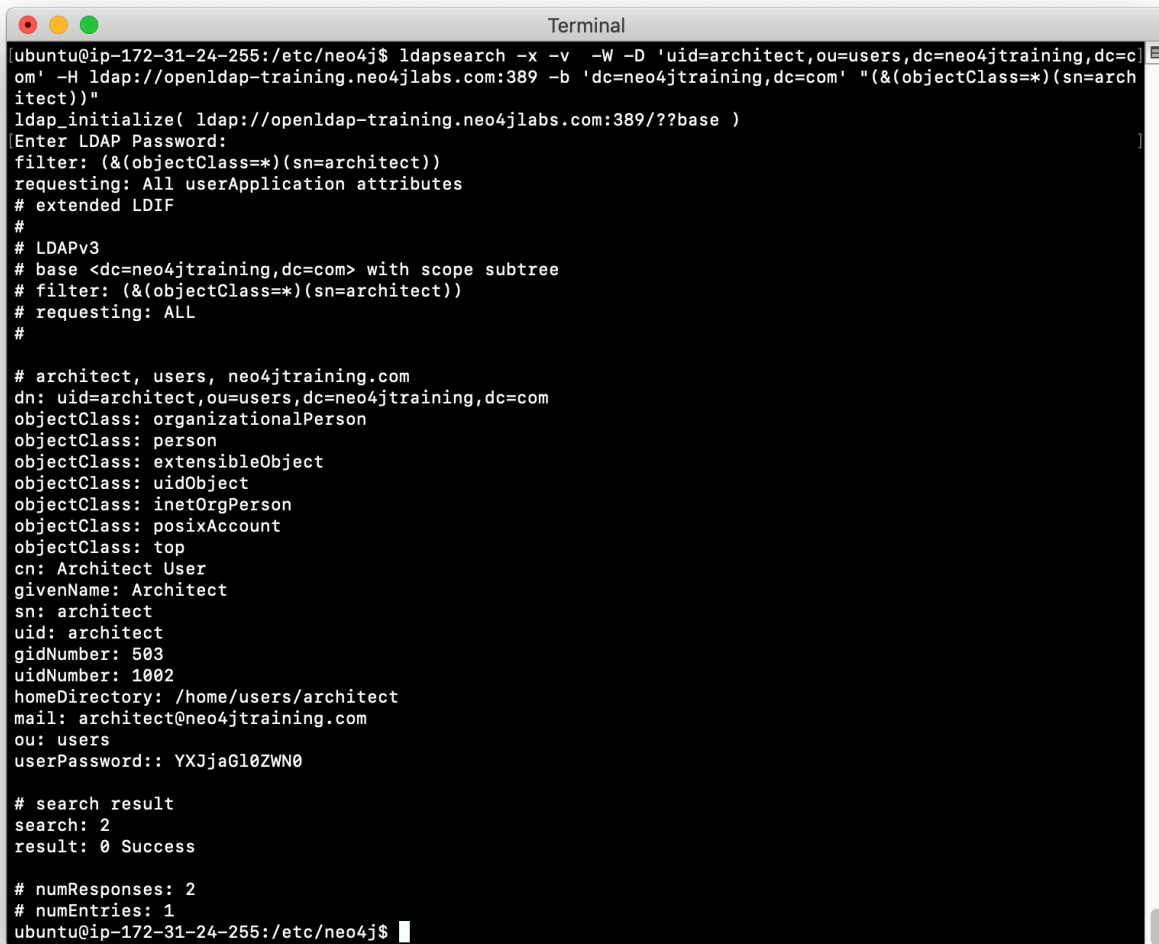
```
ldapsearch -x -v  -W -D '<designated-name-for-user>'
                     -H <ldap-server>:<port>
                     -b '<search-base>'
                     "<filter-for-user>"
```

Example: Retrieving the user, architect

Here is an example for retrieving the user *architect* in the training LDAP Server:

```
ldapsearch -x -v  -W -D 'uid=architect,ou=users,dc=neo4jtraining,dc=com'
            -H ldap://openldap-training.neo4jlabs.com:389
            -b 'dc=neo4jtraining,dc=com'
            "(&(objectClass=*)(sn=architect))"
```

Here is the result of executing this `ldapsearch` command. When you execute this type of search, you must provide the password for the user which in this LDAP is *architect*.

```
[ubuntu@ip-172-31-24-255:/etc/neo4j$ ldapsearch -x -v  -W -D 'uid=architect,ou=users,dc=neo4jtraining,dc=c]
om' -H ldap://openldap-training.neo4jlabs.com:389 -b 'dc=neo4jtraining,dc=com' "(&(objectClass=*)(sn=arch
itect))"
ldap_initialize( ldap://openldap-training.neo4jlabs.com:389/??base )
[Enter LDAP Password:                                                                                     ]
filter: (&(objectClass=*)(sn=architect))
requesting: All userApplication attributes
# extended LDIF
#
# LDAPv3
# base <dc=neo4jtraining,dc=com> with scope subtree
# filter: (&(objectClass=*)(sn=architect))
# requesting: ALL
#

# architect, users, neo4jtraining.com
dn: uid=architect,ou=users,dc=neo4jtraining,dc=com
objectClass: organizationalPerson
objectClass: person
objectClass: extensibleObject
objectClass: uidObject
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: Architect User
givenName: Architect
sn: architect
uid: architect
gidNumber: 503
uidNumber: 1002
homeDirectory: /home/users/architect
mail: architect@neo4jtraining.com
ou: users
userPassword:: YXJjaGl0ZWN0

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
ubuntu@ip-172-31-24-255:/etc/neo4j$ 
```

In this example, you see that this particular user is part of many object classes. In a large user directory, you should provide a more specific filter to improve the search.

Example: Optimized retrieval of the user, architect

For example, a user directory will contain entries for non-users such as printers. If you want to focus the search only on people, you would specify something like the following to filter by *person*:

```
ldapsearch -x -v  -W -D 'uid=architect,ou=users,dc=neo4jtraining,dc=com'
           -H ldap://openldap-training.neo4jlabs.com:389
           -b 'dc=neo4jtraining,dc=com'
           "(&(objectClass=person)(sn=architect))"
```
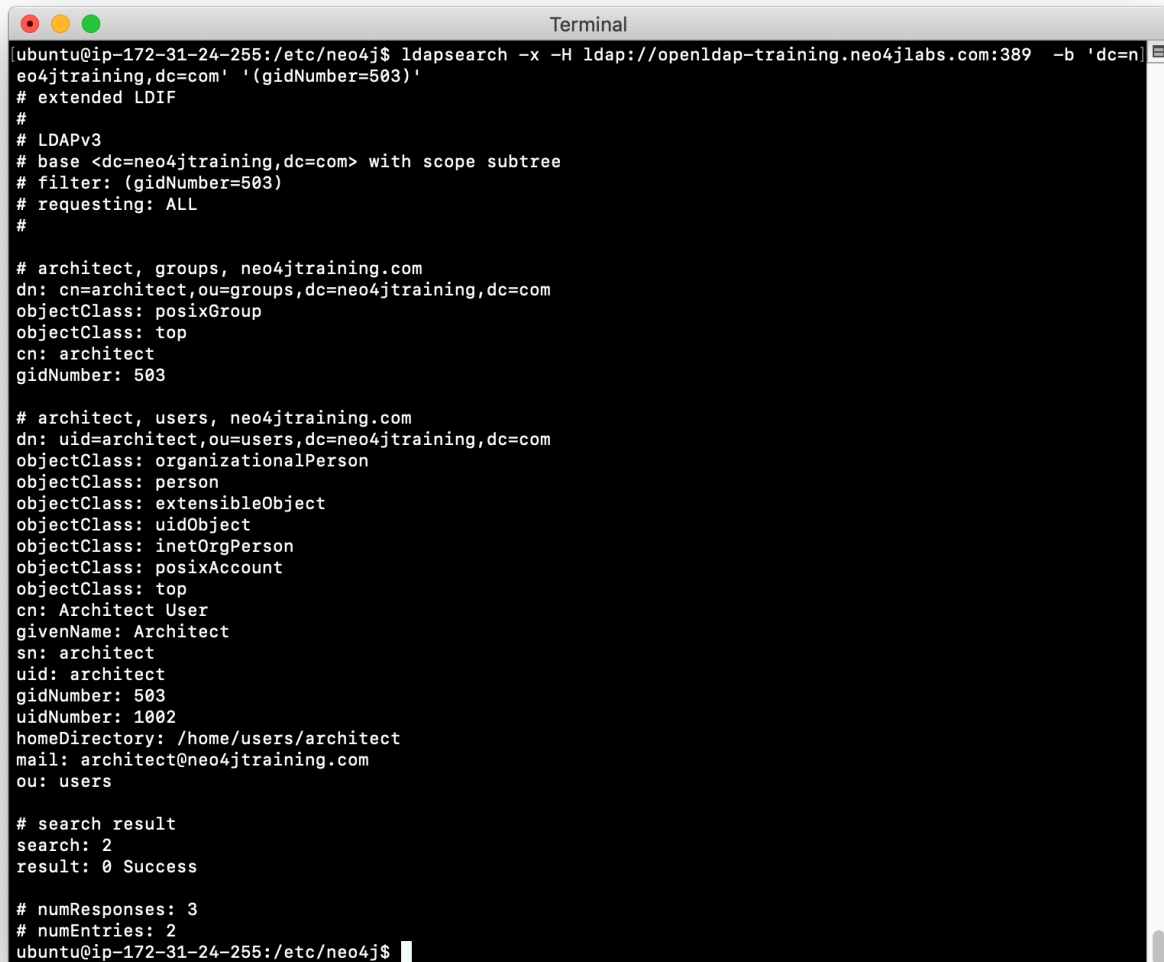
Example: Retrieving entries for a group

For integration with Neo4j, you must examine the LDAP entries to understand what groups are defined as they will need to be mapped to Neo4j native roles and possibly custom roles. In the previous image, you saw that the *gidNumber* attribute is assigned a value of 503. In LDAP, a user can be a member of more than one group.

For example, once you determine the attribute that is used to define group membership in the LDAP, you can perform a query to retrieve all entries that use that same *gidNumber* of *503*:

```
ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b
'dc=neo4jtraining,dc=com' '(gidNumber=503)'
```

And here we see an entry for a user and an entry for a group.

```
[ubuntu@ip-172-31-24-255:/etc/neo4j$ ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b 'dc=n]
eo4jtraining,dc=com' '(gidNumber=503)'
# extended LDIF
#
# LDAPv3
# base <dc=neo4jtraining,dc=com> with scope subtree
# filter: (gidNumber=503)
# requesting: ALL
#

# architect, groups, neo4jtraining.com
dn: cn=architect,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: architect
gidNumber: 503

# architect, users, neo4jtraining.com
dn: uid=architect,ou=users,dc=neo4jtraining,dc=com
objectClass: organizationalPerson
objectClass: person
objectClass: extensibleObject
objectClass: uidObject
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: Architect User
givenName: Architect
sn: architect
uid: architect
gidNumber: 503
uidNumber: 1002
homeDirectory: /home/users/architect
mail: architect@neo4jtraining.com
ou: users

# search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
ubuntu@ip-172-31-24-255:/etc/neo4j$
```

Notice that the group entry has an *objectClass* of *posixGroup*. This tells you that all groups are defined with this attribute in this particular LDAP.

Example: Retrieving group entries

So, for example, you can then do a query to find all entries that have this attribute to learn about which groups are defined in the LDAP:

```
ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b
'dc=neo4jtraining,dc=com' '(objectClass=posixGroup)'
```
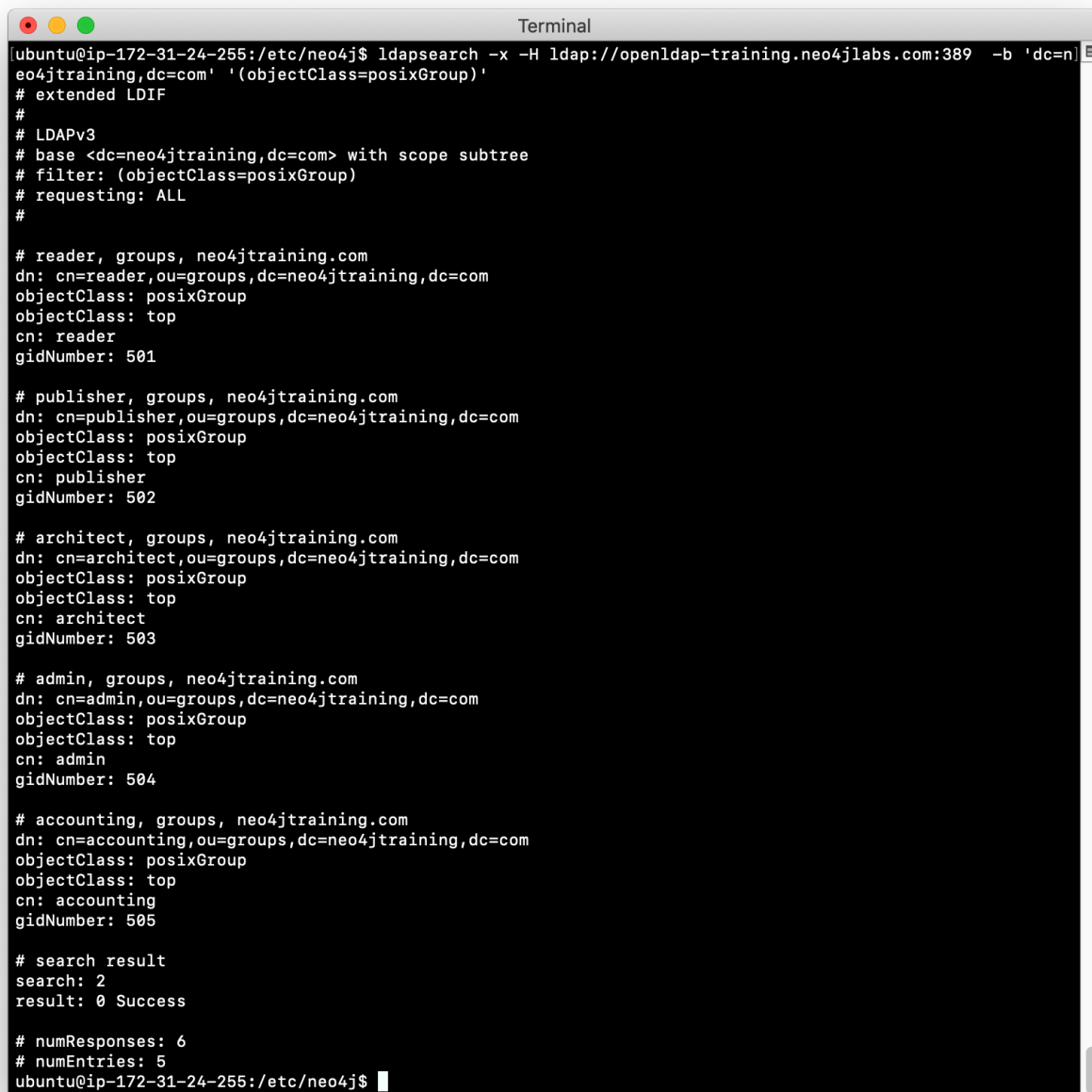
```
[ubuntu@ip-172-31-24-255:/etc/neo4j$ ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b 'dc=n]
eo4jtraining,dc=com' '(objectClass=posixGroup)'
# extended LDIF
#
# LDAPv3
# base <dc=neo4jtraining,dc=com> with scope subtree
# filter: (objectClass=posixGroup)
# requesting: ALL
#

# reader, groups, neo4jtraining.com
dn: cn=reader,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: reader
gidNumber: 501

# publisher, groups, neo4jtraining.com
dn: cn=publisher,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: publisher
gidNumber: 502

# architect, groups, neo4jtraining.com
dn: cn=architect,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: architect
gidNumber: 503

# admin, groups, neo4jtraining.com
dn: cn=admin,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: admin
gidNumber: 504

# accounting, groups, neo4jtraining.com
dn: cn=accounting,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: accounting
gidNumber: 505

# search result
search: 2
result: 0 Success

# numResponses: 6
# numEntries: 5
ubuntu@ip-172-31-24-255:/etc/neo4j$
```

What queries you perform on your LDAP provider will depend on the discovery of entries in the LDAP.

Exercise #1: Exploring the training User Directory

In this Exercise, you will simply execute a couple of commands to retrieve data from an existing LDAP server that will be used for the Exercises of this training.

Before you begin:
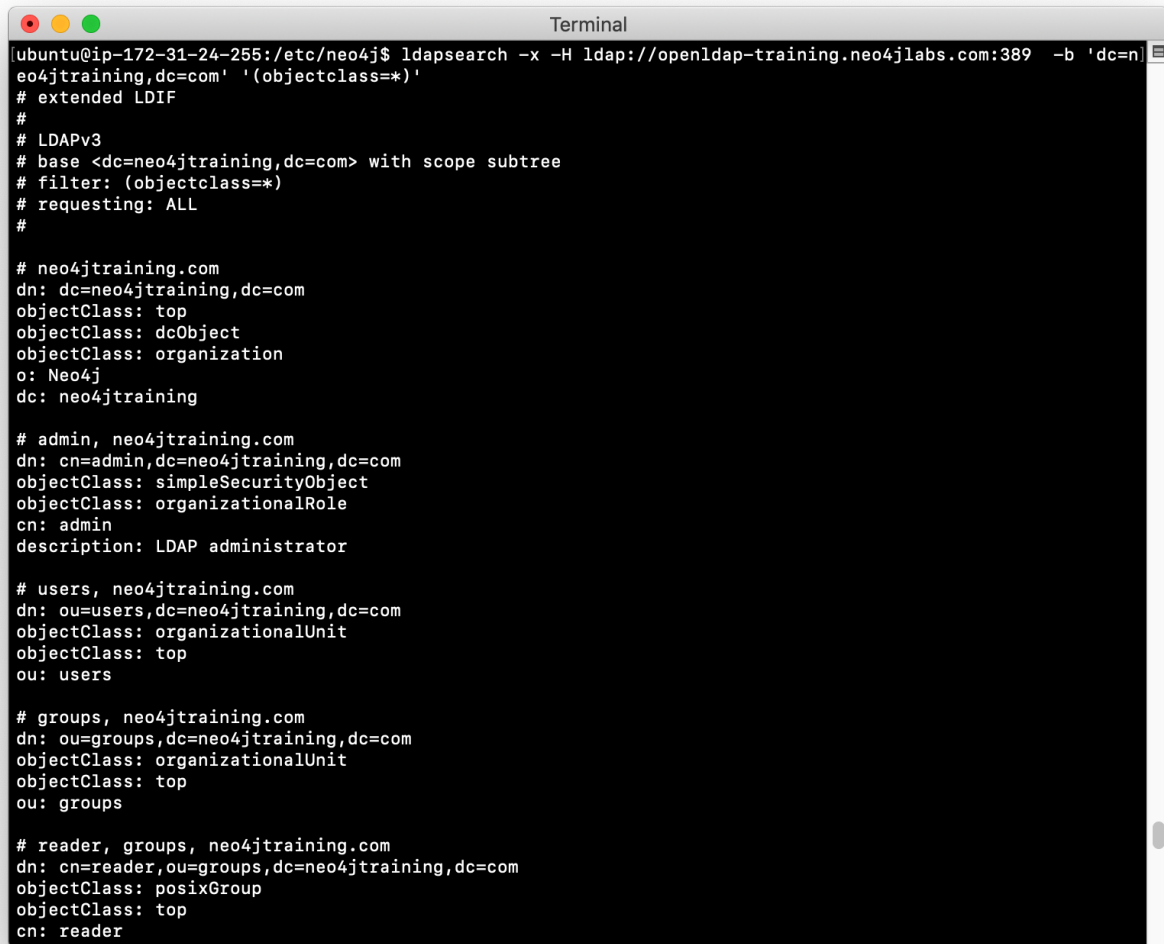
1. Open a terminal window on your system where you have worked with a stand-alone Neo4j instance in the *Managing a Neo4j Database* module.

2. Ensure that you have the LDAP utilities package installed on your system. (For example on Debian: `sudo apt-get install ldap-utils`).

Exercise steps:

In this example, and for the LDAP provider (an EC2 instance) you will be using for the Exercises in this module, the users are all part of the **neo4jtraining.com** domain.

1. Execute this `ldapsearch` command to retrieve all entries from the LDAP Server:

```
ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b
'dc=neo4jtraining,dc=com' '(objectclass=*)'
```

```
[ubuntu@ip-172-31-24-255:/etc/neo4j$ ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b 'dc=n
eo4jtraining,dc=com' '(objectclass=*)'
# extended LDIF
#
# LDAPv3
# base <dc=neo4jtraining,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# neo4jtraining.com
dn: dc=neo4jtraining,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
o: Neo4j
dc: neo4jtraining

# admin, neo4jtraining.com
dn: cn=admin,dc=neo4jtraining,dc=com
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator

# users, neo4jtraining.com
dn: ou=users,dc=neo4jtraining,dc=com
objectClass: organizationalUnit
objectClass: top
ou: users

# groups, neo4jtraining.com
dn: ou=groups,dc=neo4jtraining,dc=com
objectClass: organizationalUnit
objectClass: top
ou: groups

# reader, groups, neo4jtraining.com
dn: cn=reader,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: reader
```

NOTE
In an enterprise LDAP, you will probably not want to perform this type of query as it will return too much information, but in our training environment, you can perform this type of query to understand want is defined in our training LDAP Server.

2. Execute this `ldapsearch` command to retrieve the *reader* entry from the LDAP Server. When prompted for the password, enter *reader*.

```
ldapsearch -x -v  -W -D 'uid=reader,ou=users,dc=neo4jtraining,dc=com' -H
ldap://openldap-training.neo4jlabs.com:389  -b 'dc=neo4jtraining,dc=com'
"(&(objectClass=*)(cn=reader))" memberOf
```
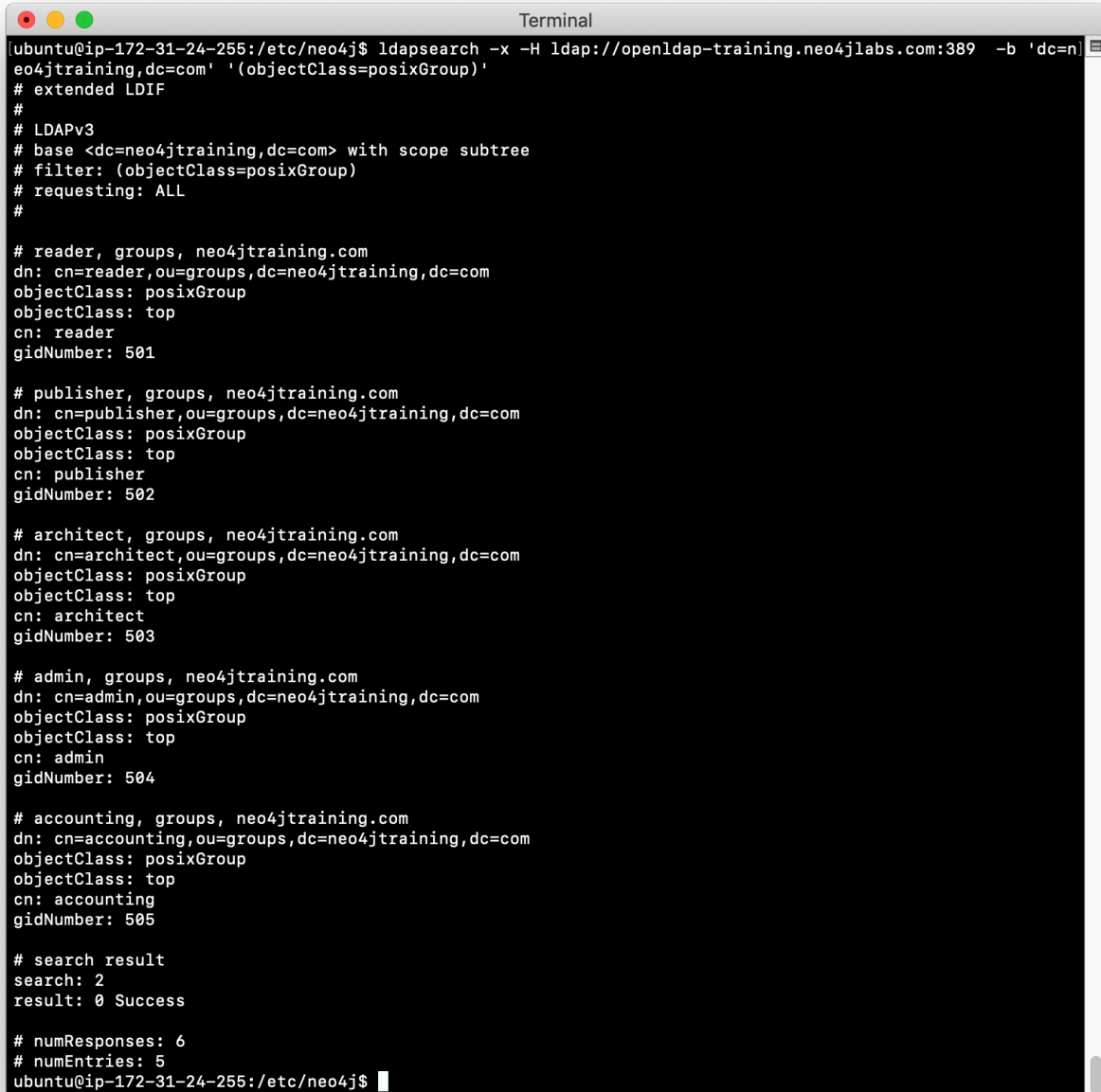
```
● ● ●                              Terminal
[ubuntu@ip-172-31-24-255:/etc/neo4j$ ldapsearch -x -v  -W -D 'uid=reader,ou=users,dc=neo4jtraining,dc=com']
 -H ldap://openldap-training.neo4jlabs.com:389  -b 'dc=neo4jtraining,dc=com' "(&(objectClass=*)(cn=reader
))" memberOf
ldap_initialize( ldap://openldap-training.neo4jlabs.com:389/??base )
[Enter LDAP Password:                                                                                    ]
filter: (&(objectClass=*)(cn=reader))
requesting: memberOf
# extended LDIF
#
# LDAPv3
# base <dc=neo4jtraining,dc=com> with scope subtree
# filter: (&(objectClass=*)(cn=reader))
# requesting: memberOf
#

# reader, groups, neo4jtraining.com
dn: cn=reader,ou=groups,dc=neo4jtraining,dc=com

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
ubuntu@ip-172-31-24-255:/etc/neo4j$ █
```

3. Execute the search for another user in the LDAP, for example *publisher*.

4. Execute the search for returning all group entries in the LDAP.

```
[ubuntu@ip-172-31-24-255:/etc/neo4j$ ldapsearch -x -H ldap://openldap-training.neo4jlabs.com:389  -b 'dc=n]
eo4jtraining,dc=com' '(objectClass=posixGroup)'
# extended LDIF
#
# LDAPv3
# base <dc=neo4jtraining,dc=com> with scope subtree
# filter: (objectClass=posixGroup)
# requesting: ALL
#

# reader, groups, neo4jtraining.com
dn: cn=reader,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: reader
gidNumber: 501

# publisher, groups, neo4jtraining.com
dn: cn=publisher,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: publisher
gidNumber: 502

# architect, groups, neo4jtraining.com
dn: cn=architect,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: architect
gidNumber: 503

# admin, groups, neo4jtraining.com
dn: cn=admin,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: admin
gidNumber: 504

# accounting, groups, neo4jtraining.com
dn: cn=accounting,ou=groups,dc=neo4jtraining,dc=com
objectClass: posixGroup
objectClass: top
cn: accounting
gidNumber: 505

# search result
search: 2
result: 0 Success

# numResponses: 6
# numEntries: 5
ubuntu@ip-172-31-24-255:/etc/neo4j$
```
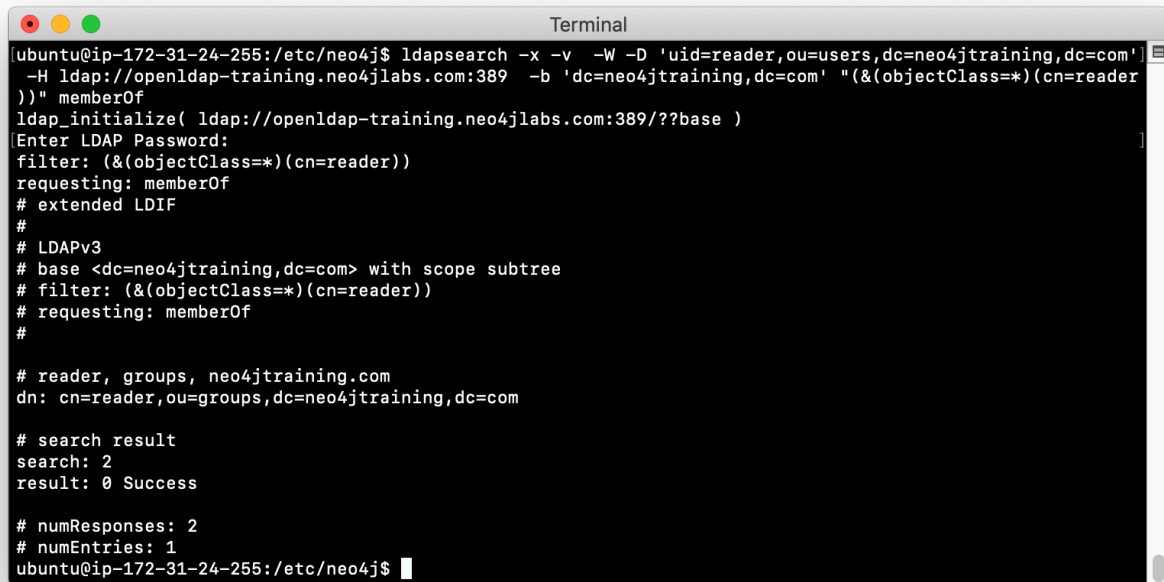
You have now confirmed that you can access the LDAP Server that will be used for authentication with the Neo4j instance and you have explored the entries in the LDAP.

Configuring Neo4j for LDAP authentication and authorization

There are many configuration settings related to authentication and authorization in the **neo4j.conf** file. The Neo4j Operations Manual describes how you can configure Neo4j to connect to and use an LDAP provider.

You can configure Neo4j to work with:

- Active Directory
- Active Directory using a sAMAccountName
- openLDAP

Each of these LDAP providers require different settings for authentication. For example using *AD* and *openLDAP*, you specify a value for *dbms.security.ldap.authentication.user_dn_template* and for *AD* with *sAMAccountName*, you specify values for other properties.

Using *AD* with *SAMAccountName* gives you the greatest flexibility because with this configuration, you do not need to specify a fixed *dn* template for authentication. Most enterprise deployments use this option.

There are several properties you specify for authorization and how you set these properties will depend on your LDAP provider.

Settings for LDAP group membership

In summary, there are three options for authorization with LDAP to establish the user's group membership:

Settings	Description
use_system_account = false	The user will search their own group membership during authentication.
use_system_account = true *use_samaccountname=false*	The system account will log in and search for the users group membership.
use_system_account = true *use_samaccountname=true*	The system account will log in and search for the user based upon their *samaccountname* and will do both group membership as well as return the users *dn* so that bind (authentication) can be completed for the user.

Configuration for openLDAP

For *openLDAP*, you set up a stand-alone or core and read replica servers in a cluster for authentication using an LDAP Server and configure the following:

```
dbms.security.auth_enabled=true
dbms.security.auth_provider=ldap
#---
dbms.security.ldap.host=<host IP address where LDAP Server runs>
#---
dbms.security.ldap.authentication.mechanism=simple
dbms.security.ldap.authentication.user_dn_template=uid={0},<top-level entity for
users>
#---
dbms.security.ldap.authorization.use_system_account=false
dbms.security.ldap.authorization.user_search_base=<top-level entity for users>
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=*)(uid={0}))
dbms.security.ldap.authorization.group_membership_attributes=<attribute-used-to-
define-groups-or-members>
dbms.security.ldap.authorization.group_to_role_mapping=\
    <group-or-member >  = reader; \
    <group-or-member >  = publisher; \
    <group-or-member >  = architect; \
    <group-or-member >  = admin; \
    <group-or-member > = <custom-role>
```

Example: Training configuration

For example, with the LDAP Server that you will be using in this training, you specify:

```
dbms.security.auth_enabled=true
dbms.security.auth_provider=ldap
#---
#LDAP Server running as EC2 instance
dbms.security.ldap.host=openldap-training.neo4jlabs.com
#---
dbms.security.ldap.authentication.mechanism=simple
# users are defined under ou=users,dc=neo4jtraining,dc=com in LDAP
dbms.security.ldap.authentication.user_dn_template=uid={0},ou=users,dc=neo4jtraining,d
c=com
#---
dbms.security.ldap.authorization.use_system_account=false
# limit where the search starts so entire LDAP is not searched
dbms.security.ldap.authorization.user_search_base=ou=users,dc=neo4jtraining,dc=com
# limit the entries that are searched to be people
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=person)(uid={0}))
# in this LDAP the gidnumber attribute is used to define groups
dbms.security.ldap.authorization.group_membership_attributes=gidnumber
# each group is mapped to a Neo4j native role and to the custom accounting role
dbms.security.ldap.authorization.group_to_role_mapping=\
    501  = reader; \
    502  = publisher; \
    503  = architect; \
    504  = admin; \
    505 = accounting
```

NOTE	Just as the LDAP can be configured to allow users to be members of multiple groups. You can also specify multiple groups to be mapped to the same role.

After you have made these configuration changes for the Neo4j instance, you restart the instance and then test that users in the LDAP can access the Neo4j instance.

The **security.log** file contains log records of all users that connected to or attempted to connect to the Neo4j instance. Later in this training, you will learn more about monitoring and logging. If you need to troubleshoot an authentication issue, you can set *dbms.log.security.level=DEBUG* in your Neo4j configuration to see more information about the login attempts. In most cases the troubleshooting will need to occur on the LDAP Server side to better understand the requests received from the Neo4j instance.

Multiple authentication providers

It is possible to use native and another authentication provider together in situations where a small number of users would be maintaining the database, but most of the users would use the enterprise authentication provider such as LDAP. To to this, your configuration setting would be:

```
dbms.security.auth_providers=native,ldap
```

Turning off role mapping (temporary)

In a troubleshooting situation, you may need to give all users access to the database. You can start the Neo4j instance where all role mapping is bypassed. In this example, all groups with the ID of user would have the reader role:

```
dbms.security.ldap.authorization.group_membership_attributes=objectClass
dbms.security.ldap.authorization.group_to_role_mapping="user" = reader
```

In the next Exercise, you will configure a Neo4j instance to use authentication with the LDAP Server you accessed in the previous exercise.

Exercise #2: Configuring authentication for a stand-alone instance

In this Exercise, you will modify the configuration for the stand-alone Neo4j instance that you have worked with in the *Managing a Neo4j Database* module, prior to using Docker for Causal Clustering. You will use the **movies3.db** that you worked with previously with the Neo4j stand-alone instance. Then you will test that the authentication is working.
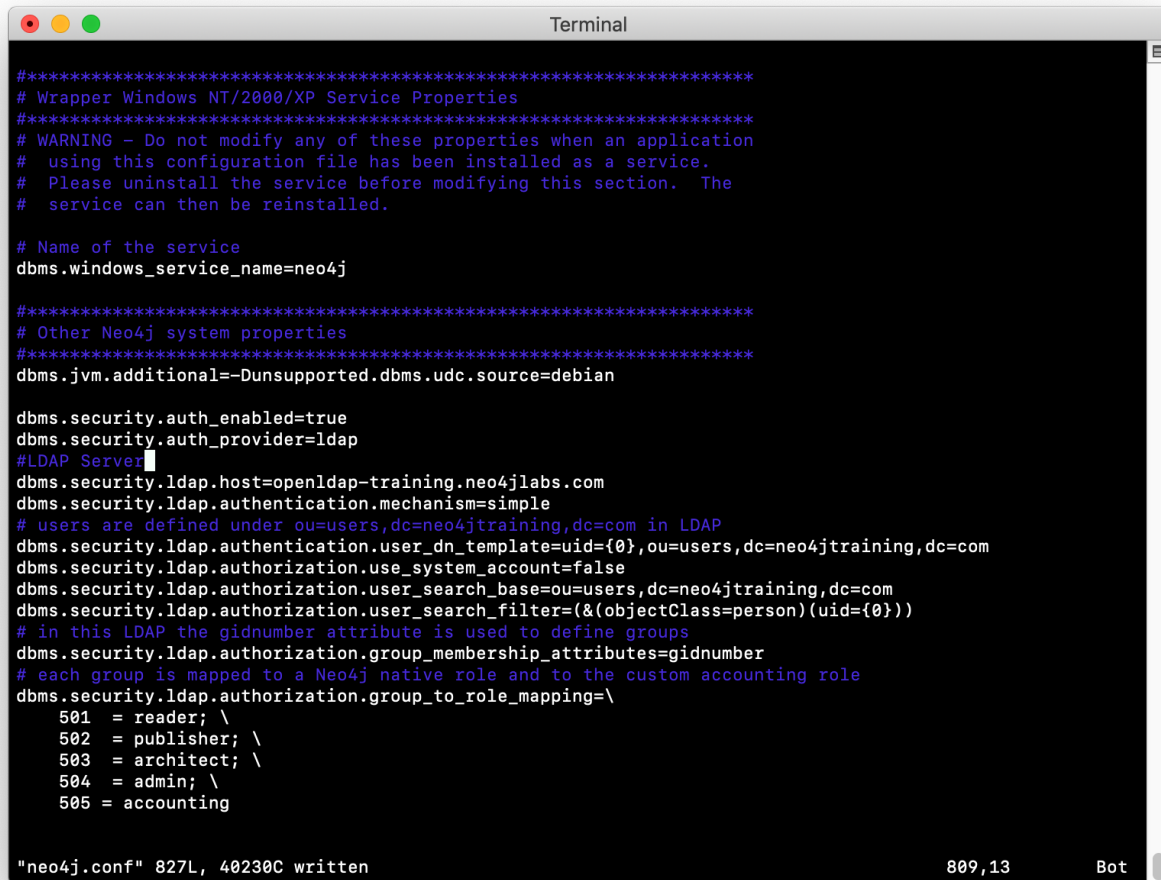
Before you begin

1. Make sure you have completed Exercise 1 that confirms that you can access the LDAP Server.

2. Modify **neo4j.conf** to use **movie3.db** as the active database.

3. Start or restart the Neo4j instance and confirm that it starts without error.

4. Open a terminal window on your system where you have worked with a stand-alone Neo4j instance using the **movie3.db** database.

5. Stop the Neo4j instance.

Exercise steps:

1. Make a copy of **neo4j.conf**, which is the last good configuration for this Neo4j instance.

2. Modify the properties in this file to use the training LDAP Server as the LDAP provider (openldap-training.neo4jlabs.com). You can set the properties in their locations in the **neo4j.conf** file, but a useful way to work is to add the properties you are setting to the end of the file so you can see all of your modifications in one place.

```
#*******************************************************************
# Wrapper Windows NT/2000/XP Service Properties
#*******************************************************************
# WARNING - Do not modify any of these properties when an application
#  using this configuration file has been installed as a service.
#  Please uninstall the service before modifying this section.  The
#  service can then be reinstalled.

# Name of the service
dbms.windows_service_name=neo4j

#*******************************************************************
# Other Neo4j system properties
#*******************************************************************
dbms.jvm.additional=-Dunsupported.dbms.udc.source=debian

dbms.security.auth_enabled=true
dbms.security.auth_provider=ldap
#LDAP Server
dbms.security.ldap.host=openldap-training.neo4jlabs.com
dbms.security.ldap.authentication.mechanism=simple
# users are defined under ou=users,dc=neo4jtraining,dc=com in LDAP
dbms.security.ldap.authentication.user_dn_template=uid={0},ou=users,dc=neo4jtraining,dc=com
dbms.security.ldap.authorization.use_system_account=false
dbms.security.ldap.authorization.user_search_base=ou=users,dc=neo4jtraining,dc=com
dbms.security.ldap.authorization.user_search_filter=(&(objectClass=person)(uid={0}))
# in this LDAP the gidnumber attribute is used to define groups
dbms.security.ldap.authorization.group_membership_attributes=gidnumber
# each group is mapped to a Neo4j native role and to the custom accounting role
dbms.security.ldap.authorization.group_to_role_mapping=\
    501  = reader; \
    502  = publisher; \
    503  = architect; \
    504  = admin; \
    505 = accounting


"neo4j.conf" 827L, 40230C written                         809,13       Bot
```

3. Start the Neo4j instance.

4. Examine the log file to ensure that the instance started without errors. If it does not start, review/adjust the properties you set in **neo4j.conf**.

5. Start cypher-shell specifying the user *reader/reader*. Can you log in?

6. Enter a Cypher statement that reads from the database `MATCH (n) RETURN count(n);`

7. Enter a Cypher statement that writes to the database `CREATE (p:Person {name:'John'}) RETURN p.name;` Did you see an error? This is because a user with the *reader* role cannot modify the database.

```
● ● ●                        ubuntu@ip-172-31-24-255: ~
[ubuntu@ip-172-31-24-255:~$ /usr/bin/cypher-shell -u reader -p reader
Connected to Neo4j 3.5.1 at bolt://localhost:7687 as user reader.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
[neo4j> MATCH (n) RETURN count(n);
+----------+
| count(n) |
+----------+
| 4193782  |
+----------+

1 row available after 37 ms, consumed after another 1 ms
[neo4j> CREATE (p:Person {name:'John'}) RETURN p.name;
Token create operations are not allowed for user 'reader' with roles [reader].
neo4j>
```

8. Exit out of cypher-shell.

9. Start cypher-shell specifying the user *publisher/publisher*. Can you log in?

10. Enter a Cypher statement that reads from the database `MATCH (n) RETURN count(n);`

11. Enter a Cypher statement that writes to the database `CREATE (p:Person {name:'John'}) RETURN p.name;`

```
● ● ●                        ubuntu@ip-172-31-24-255: ~
[ubuntu@ip-172-31-24-255:~$ /usr/bin/cypher-shell -u publisher -p publisher
Connected to Neo4j 3.5.1 at bolt://localhost:7687 as user publisher.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
[neo4j> MATCH (n) RETURN count(n);
+----------+
| count(n) |
+----------+
| 4193782  |
+----------+

1 row available after 1 ms, consumed after another 0 ms
[neo4j> CREATE (p:Person {name:'John'}) RETURN p.name;
+--------+
| p.name |
+--------+
| "John" |
+--------+

1 row available after 112 ms, consumed after another 1 ms
Added 1 nodes, Set 1 properties, Added 1 labels
[neo4j> :exit

Bye!
ubuntu@ip-172-31-24-255:~$
```

12. Look at the log records in the **security.log** file. Do they correspond to your activities against the Neo4j instance?

Using LDAP for authentication, you can control which users have different types of access to the database based upon the groups defined in the LDAP. If you need to perform administrative operations against the database, you would log in as *admin/admin* which is what currently is defined for this LDAP Server used for training.

Exercise 2: Taking if further:

Configure and test the core and read replica servers you used for Causal Clustering in the previous module for authentication using the same LDAP Server.

LDAP caching

By default, the Neo4j instance caches security-related interactions between a client and the LDAP Server as an optimization. A best practice is to leave this setting. If you set it to false, the client experience slower as the authentication/authorization needs to be done with every request to the Neo4j instance. See the documentation about the *dbms.security.ldap.authentication.cache_enabled* settings and how to clear the Auth cache.

Using secure LDAP

If the LDAP provider uses encryption (LDAPS or StartTLS=true), then you must set some additional properties in **neo4j.conf** to specify that encryption will be used and what port to use. In addition, the LDAP Server Certificate needs to be added to the JVM keystore for each Neo4j instance:

```
keytool -import -alias <alias-name> -keystore ..\lib\security\cacerts -file <path-to-cert-file>
```

This is commonly used with Active Directory. See the documentation about encrypting with your particular authentication provider.

All production environments should use an SSL certificate issued by a Certificate Authority when accessing their LDAP provider. You will work with the system administrator responsible for providing the certificate. You can, however, set up and configure a self-signed test certificate while you are setting up your Neo4j Instance to use the LDAP provider. To do this you specify a value for *dbms.jvm.additional* in **neo4j.conf** as specified in the documentation. Once all of your tests are complete, you can switch to the real SSL certificate issued by the Certificate Authority for the LDAP Server.

Securing data-in-transit

Users "in the building", such as employees can access the application from an internal network. These networks are secure. In addition, "off site" employees typically access the application using a Virtual Private Network (VPN) which is also secure. Customers or other users who access the application over the Internet, by default, do not use a secure connection. If these customers send private data like pass-phrases and credit card numbers, it can be examined, stolen, or altered by a malicious actor somewhere on the Internet. To secure their connection, you must provide SSL/TLS access your application. SSL/TLS is a software layer that encrypts data sent and received over the Internet.

How to secure data-in-transit

If your application has users that access the application over the Internet, you should implement SSL/TLS by doing the following.

- Identify which server processes are accessed by users over the Internet.

- Identify server ports that will be used for these server processes.

- Configure each server process to use SSL ports.

- Create and publish secure digital certificates for each server process.

A digital certificate is used by client applications when they communicate with the server processes using SSL/TLS. A digital certificate must have an expiration date.

NOTE	You must ensure that the version of SSL/TLS software your application uses is up-to-date and that it supports all the types of clients that will be accessing your application. For example, not all browsers are compatible with the latest SSL/TLS versions.

Recommendations for external users

You should recommend that external users store the following artifacts securely:

- Digital certificates

- User credentials (encrypt if they are included in code or a configuration file; never use clear-text credentials)

Configuring the Neo4j instance for SSL/TLS

Because the use of certificates is secure and requires a domain that you own as well as certificates from a trusted Certificate Authority (CA), you will learn how to configure a Neo4j instance, but will not perform the steps in your training environment. The creation of certificates and keys should be used in your real production environment.

Here are the steps you should take for your production system.

Step 1: Install JCE

1. To use SSL/TLS with the Neo4j instance, you must first ensure that the Java Cryptography Extension (JCE) is installed on the host machine.

2. Download the JCE zip file from the above location or from here

3. Unzip the file to create **local_policy.jar** and **US_export_policy.jar**.

4. Navigate to the Java 8 **jre/lib/security/policy/unlimited** folder.

5. Save a copy of the existing files.

6. Copy the two **.jar** files you unzipped into this **unlimited** folder.

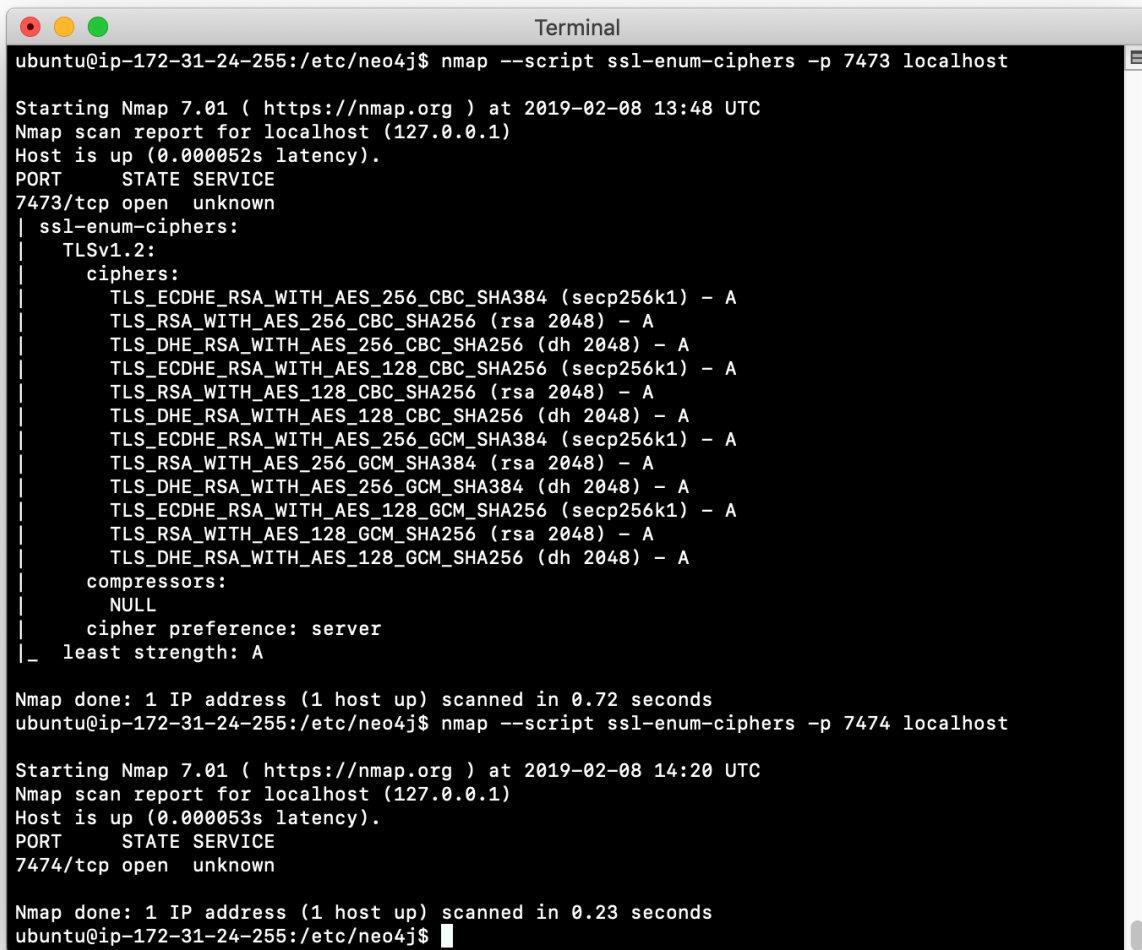## Confirming ports are SSL-enabled

Your Neo4j instance can only provide SSL if you have installed JCE. On Linux/Debian, you can confirm that your specific ports of the running Neo4j instance are SSL-capable by executing one of the following commands:

```
# a causal cluster raft listen address
nmap --script ssl-enum-ciphers -p 7000 localhost
# for HTTPS
nmap --script ssl-enum-ciphers -p 7473 localhost
```

Example: Ports used

Suppose we have installed JCE for use with our Neo4j instance. Here is an example where we see that the HTTPS port uses these ciphers and the HTTP port does not:

```
ubuntu@ip-172-31-24-255:/etc/neo4j$ nmap --script ssl-enum-ciphers -p 7473 localhost

Starting Nmap 7.01 ( https://nmap.org ) at 2019-02-08 13:48 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000052s latency).
PORT      STATE SERVICE
7473/tcp open  unknown
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (secp256k1) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA256 (rsa 2048) - A
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (dh 2048) - A
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (secp256k1) - A
|       TLS_RSA_WITH_AES_128_CBC_SHA256 (rsa 2048) - A
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (dh 2048) - A
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256k1) - A
|       TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A
|       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (dh 2048) - A
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secp256k1) - A
|       TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 2048) - A
|       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (dh 2048) - A
|     compressors:
|       NULL
|     cipher preference: server
|_  least strength: A

Nmap done: 1 IP address (1 host up) scanned in 0.72 seconds
ubuntu@ip-172-31-24-255:/etc/neo4j$ nmap --script ssl-enum-ciphers -p 7474 localhost

Starting Nmap 7.01 ( https://nmap.org ) at 2019-02-08 14:20 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000053s latency).
PORT      STATE SERVICE
7474/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.23 seconds
ubuntu@ip-172-31-24-255:/etc/neo4j$
```

Step 2: Obtain digital certificates

Determine how many security policies you will use. For example, you may have policies for clients, backups, and cluster communications. Each policy will have its own private key (**private.key**) and public certificate (**public.crt**). You must obtain the certificates from a trusted Certificate Authority in PEM format. If your application requires multiple certificates, they can be combined into a single file.

To obtain a certificate you must first have a domain name that you own. You can buy a fixed domain name or you can buy a more flexible one from a site such as dyn.com. There are many providers of certificates you can use. Here is an article that explains how to use LetsEncrypt which works well with Neo4j instances.

Step 3: Create folders for the security policies

The default location for certificates is defined in **neo4j.conf** with the *dbms.directories.certificates* property.

In this location, create a folder for each security policy your application requires, for example **client_policy**, **backup_policy**, **cluster_policy**. Then, in each of these folders, create the **revoked** and **trusted** sub-folders.

Step 4: Place keys and certificates into your Neo4j installation

Place the **.key** and **.cert** files in the top-level folder for the security policy. Place a copy of the **.cert** file in the **trusted** folder for the security policy.

NOTE	If you are using *LetsEncrypt*, follow the instructions in the article regarding links to the certificates and keys rather than copying them.

Step 5: Configure SSL for the Neo4j instance

Here is an example of recommended settings to configure SSL for a Neo4j instance that will support SSL for all clients:

```
# client policy
dbms.ssl.policy.client_policy.base_directory=/var/lib/neo4j/certificates/client_policy
dbms.ssl.policy.client_policy.private_key=/var/lib/neo4j/certificates/client_policy/pr
ivate.key
dbms.ssl.policy.client_policy.public_certificate=/var/lib/neo4j/certificates/client_po
licy/public.cert
dbms.ssl.policy.client_policy.client_auth=REQUIRE
bolt.ssl_policy=client_policy
https.ssl_policy=client_policy

#all bolt communication must be encrypted
dbms_connector_bolt_tls_level=REQUIRED

#prevent use of http port
dbms.connector.http.enabled=false

# Web access only using HTTPS for Neo4j Browser
dbms.security.http_strict_transport_security=Strict-TransportSecurity: max-
age=31536000; includeSubDomains
```

The directory **client_policy** is where the certificate and key is available to the Neo4j instance. All bolt communication must use *tls_level*, which ensures that all credentials used for authentication and clear-text data are encrypted. Additionally, we ensure that all Web access for the Neo4j Browser must be via HTTPS.

SSL for Causal Clusters

For each Neo4j instance that will serve as a core server or read replica, you should configure SSL as you do for a stand-alone Neo4j instance. You must configure a separate policy for the cluster, for example *cluster_policy*. Each server will use both the *client_policy* and the *cluster_policy*.

Here is an example of the SSL configuration for a member of a cluster that you add to the Neo4j configuration in addition to the SSL configuration settings you saw earlier for the client policy:

```
# cluster policy
dbms.ssl.policy.cluster_policy.base_directory=/var/lib/neo4j/certificates/cluster_poli
cy
dbms.ssl.policy.cluster_policy.private_key=/var/lib/neo4j/certificates/cluster_policy/
private.key
dbms.ssl.policy.cluster_policy.public_certificate=/var/lib/neo4j/certificates/cluster_
policy/public.cert
dbms.ssl.policy.cluster_policy.client_auth=REQUIRE

# specify TLS version
dbms.ssl.policy.cluster_policy.tls_version=TLSv1.2

# specify ciphers used
dbms.ssl.policy.cluster_policy.ciphers=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

causal_clustering.ssl_policy=cluster_policy

# use a specfic backup policy for the cluster
dbms.backup.ssl_policy=backup_policy
```

SSL for backups

Whether you are using Neo4j stand-alone or in a cluster, a best practice is to also ensure that SSL is used for backups. To do so, you can create and configure a policy, for example *backup_policy*, for backups that also requires a certificate.

Additional settings related to data-in-transit

Here are some additional settings that you should consider to help reduce unintended access to your Neo4j instances:

```
#prevent browser from caching logging information
browser.retain_connection_credentials=false

#prevent browser staying open
browser.credential_timeout=15m

# Using a specific CORS response header
dbms.connectors.access_control_allow_origin=neo4j.com

# disable data usage collection by third parties
dbms.udc.enabled=false
```

Best practices for developers

You should work with your security administrator to come up with a set of guidelines for developers of your application to ensure that data-in-transit is secure. For example, developers should never include private data in a Cypher statement, but use parameters to avoid Cypher injections at runtime.

Securing data-at-rest

Securing data-at-rest means securing private data in a Neo4j installation including databases and backups.

An enterprise database typically contains private customer, personal, and financial data. If you do not secure the private data in the database, a malicious actor could access or steal the database and the private data would be exposed.

Securing files at the OS level

OS-level security restricts access to application files such as:

- Database files
- Application-related files outside the database
- Configuration files for the Neo4j instance
- Neo4j executables
- Application executables and libraries
- Security-related files (keys, digital certificates)

The system administrator is responsible for setting the access permissions for all files stored in the filesystem. If a malicious actor can access a directory in a filesystem that contains application files, they can steal, modify, or delete files in that directory. In addition, if the files in that directory contain private information, this information could be exposed.

Ensure that Neo4j does not run as root

Using a non-privileged, dedicated service account restricts the database from accessing the critical areas of the operating system that are not required by the Neo4j instance. This will also mitigate the potential for unauthorized access via a compromised, privileged account on the operating system.

You can determine which processes are owned by *neo4j* using: `ps -ef |grep -E "neo4j"`

By default the user, *neo4j* owns the processes running that are related to the Neo4j instance.

Neo4j file permissions

The following directories should be read-only:

- **conf**
- **import**
- **bin**
- **lib**
- **plugins**

The following directories should be read/write:

- **data**
- **logs**
- **metrics**

And the **bin** directory should be execute.

You should also set the log files to only be writable by *neo4j* and readable by *root*.

Security auditing

You must have a plan for analyzing a suspected security breach so you can respond quickly and effectively. As part of that plan, you should implement a security audit trail that captures which users logged in successfully and those that failed.

Implementing an audit trail does not secure your application against malicious actors, but the information that is captured can be used to alert you of a potential security breach or to aid in the investigation of a suspected security breach.

Here are some recommended settings for security auditing in Neo4j:

```
dbms.directories.logs=logs
# Log level for the security log. One of DEBUG, INFO, WARN and ERROR.
dbms.logs.security.level=INFO
# Threshold for rotation of the security log.
dbms.logs.security.rotation.size=20m
# Minimum time interval after last rotation of the security log before
it may be rotated again.
dbms.logs.security.rotation.delay=300s
# Maximum number of history files for the security log.
dbms.logs.security.rotation.keep_number=7
```

In a troubleshooting situation or if you want to more closely monitor access, you would set the level to *DEBUG*.
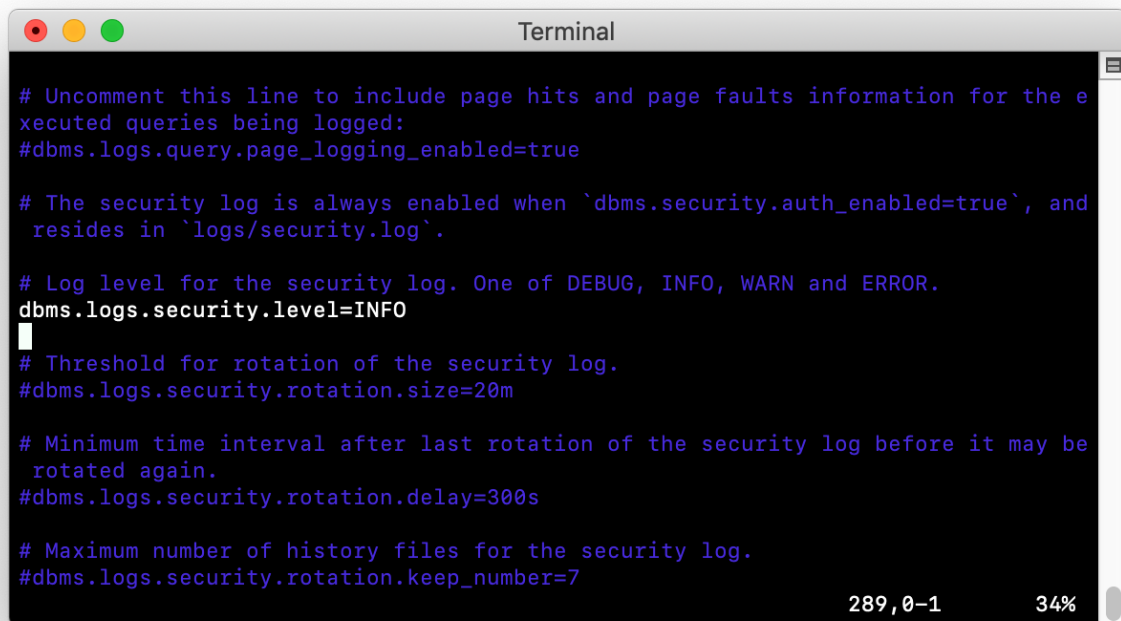
Exercise #3: Security auditing

In this Exercise, you will modify the security auditing configuration for the stand-alone Neo4j instance that you have worked earlier in this module.

Before you begin

1. Make sure you have completed Exercise 2 where you have configured and tested the Neo4j instance to use the training LDAP Server.

2. Open a terminal window on your system.

3. Stop the Neo4j instance.

Exercise steps:

1. Make a copy of **neo4j.conf**, which is the last good configuration for this Neo4j instance.

2. Modify the property in this file log security events at the INFO level.

```
# Uncomment this line to include page hits and page faults information for the e
xecuted queries being logged:
#dbms.logs.query.page_logging_enabled=true

# The security log is always enabled when `dbms.security.auth_enabled=true`, and
 resides in `logs/security.log`.

# Log level for the security log. One of DEBUG, INFO, WARN and ERROR.
dbms.logs.security.level=INFO

# Threshold for rotation of the security log.
#dbms.logs.security.rotation.size=20m

# Minimum time interval after last rotation of the security log before it may be
 rotated again.
#dbms.logs.security.rotation.delay=300s

# Maximum number of history files for the security log.
#dbms.logs.security.rotation.keep_number=7
                                                    289,0-1       34%
```
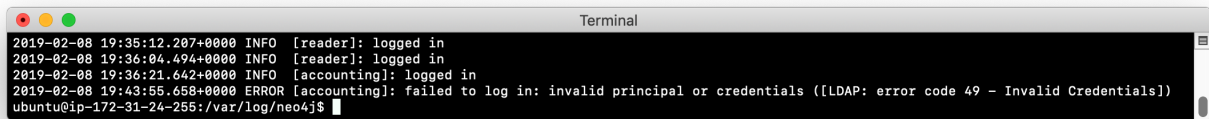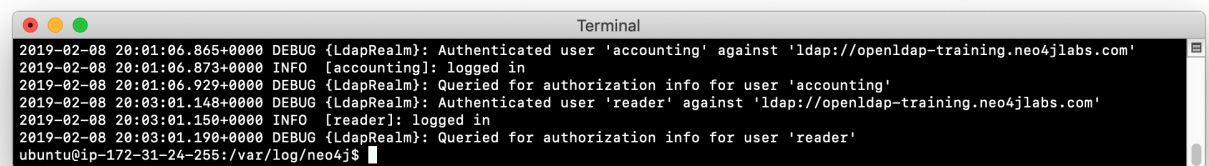
3. Start the Neo4j instance.

4. Examine the log file to ensure that the instance started without errors. If it does not start, review/adjust the properties you set in **neo4j.conf**.

5. Start cypher-shell specifying the user *reader/reader*.

6. Exit out of cypher-shell and log in again specifying *accounting/accounting*.

7. Exit out of cypher-shell and log in again specifying *accounting/foo*.

8. View the **security.log** file. Do you see these logins and the failed login attempt? The default level of logging is *INFO* if you do not set the property in the configuration file.

```
Terminal
2019-02-08 19:35:12.207+0000 INFO  [reader]: logged in
2019-02-08 19:36:04.494+0000 INFO  [reader]: logged in
2019-02-08 19:36:21.642+0000 INFO  [accounting]: logged in
2019-02-08 19:43:55.658+0000 ERROR [accounting]: failed to log in: invalid principal or credentials ([LDAP: error code 49 – Invalid Credentials])
ubuntu@ip-172-31-24-255:/var/log/neo4j$
```

9. Modify the configuration for the security level to be DEBUG.

10. Restart the Neo4j instance.

11. Start cypher-shell specifying the user *accounting/accounting*.

12. Enter a statement that reads from the database, for example: `MATCH (n) RETURN count(n);`. You should receive an error because this user does not have the *reader* role.

13. View the log file. There is information written stating that the read access was checked against the LDAP. Even though access was denied, this information is not written to the log file.

14. Exit out of cypher-shell.

15. Start cypher-shell specifying the user *reader/reader*.

16. Enter a statement that reads from the database, for example: `MATCH (n) RETURN count(n);`.

17. View the log file. Again, you should see that the instance contacted the LDAP Server for role information.


```
Terminal
2019-02-08 20:01:06.865+0000 DEBUG {LdapRealm}: Authenticated user 'accounting' against 'ldap://openldap-training.neo4jlabs.com'
2019-02-08 20:01:06.873+0000 INFO  [accounting]: logged in
2019-02-08 20:01:06.929+0000 DEBUG {LdapRealm}: Queried for authorization info for user 'accounting'
2019-02-08 20:03:01.148+0000 DEBUG {LdapRealm}: Authenticated user 'reader' against 'ldap://openldap-training.neo4jlabs.com'
2019-02-08 20:03:01.150+0000 INFO  [reader]: logged in
2019-02-08 20:03:01.190+0000 DEBUG {LdapRealm}: Queried for authorization info for user 'reader'
ubuntu@ip-172-31-24-255:/var/log/neo4j$
```

18. Exit out of cypher-shell.

19. Stop the Neo4j instance.

So as you have seen, you can audit for logins and failed logins, even with the default security level of *INFO*.

Additional security measures

You may want to consider not using any default ports for your Neo4j instances. Hackers frequently scan IP addresses for commonly used ports, so it's not uncommon to use a different port to "fly under the radar".

The `neo4j-shell` utility has been deprecated and you should not allow any users to use it. By default, it is not enabled in a Neo4j instance.

JMX is sometimes used for monitoring system activity. JMX is not secure and by default is disabled in Neo4j.

You should work closely with Neo4j Professional Services and Technical Support to ensure that your production system is secure. See this Security Checklist will also help you to determine if you have set up your systems correctly.

Check your understanding

Question 1

Suppose you will be using an enterprise LDAP provider and you must work with the LDAP administrator to ensure that all users who will be accessing the Neo4j database will have read or write access. What needs to be added to the LDAP?

Select the correct answer.

☐ The Neo4j plugin for LDAP

☐ The Neo4j LDIF file that is installed with Neo4j

☐ New user entries with the *neo4j* attribute defined of with a value of *reader* or *writer*

☐ New group entries for neo4j roles and modifications to existing user entries to be a group member

Question 2

Suppose that you want to ensure that all data in or out of a Neo4j instance is secure (data-in-transit). What must you configure?

Select the correct answers.

☐ SSL policy

☐ Enable bolt to use TLS

☐ Restrict HTTP port

☐ Require HTTPS for Neo4j Browser

Question 3

In order to configure a security policy for SSL in Neo4j, what must you obtain?

Select the correct answers.

☐ Encrypted pass-phrase file to use from Neo4j support

☐ Digital certificate from a Certificate Authority (public.crt file)

☐ SSL library (.jar) from the Certificate Authority

☐ Private key from the Certificate Authority (private.key)

Summary

You should now be able to:

- Describe what security means for an application.

- Configure a Neo4j instance to use LDAP as the authentication provider.

- Describe how to secure data-in-transit.

- Describe how to secure data-at-rest.

- Configure and use security auditing.