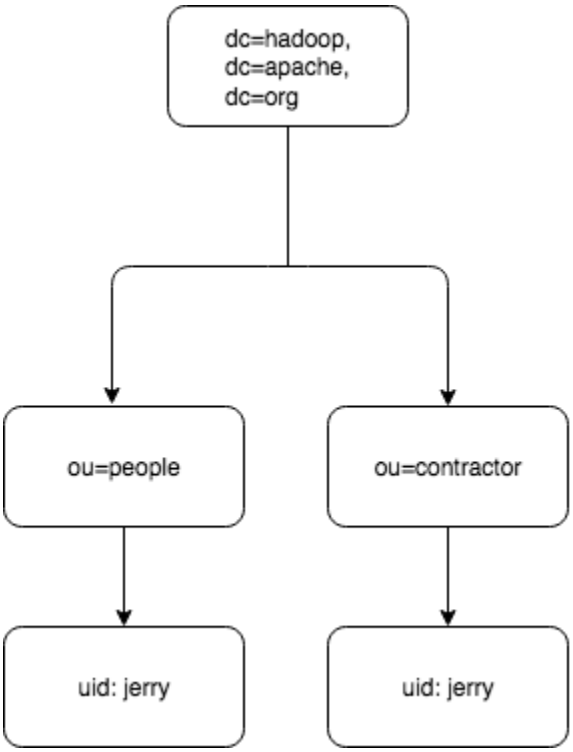
Apache Knox using multiple LDAP Realms

In large enterprise LDAP setups there could be cases where users under different OUs might have same userids for e.g.



In the diagram above we can see that we have the same userid in multiple OUs, i.e. *uid=jerry,ou=people,dc=hadoop,dc=apache,dc=org* and *uid=jerry,ou=contractor, dc=hadoop,dc=apache,dc=org.*

What if we would like both the users to successfully authenticate using Apache Knox ? one thing we could do is broaden the search base by going high up the LDAP tree, in this case using *dc=hadoop,dc=apache,dc=org* as the search base.

e.g.

```
                <param>
            <name>main.ldapRealm.searchBase</name>
            <value>dc=hadoop,dc=apache,dc=org</value>
        </param>
```

Unfortunately, this approach will work **only** for one user and not the other. The reason for this is because Knox walks down one branch and tries to authenticate and if it fails to authenticate (say, because of a bad password) it will report a failure and stop, throwing an error "*Failed to Authenticate with LDAP server: {1}*".

The issue here is that after failure while traversing down one branch (say, *ou=people,dc=hadoop,dc=apache,dc=org*) the other branch (*ou=contractor, dc=hadoop,dc=apache,dc=org*) is ignored.

The solution is to use multiple LDAP realms. Multiple LDAP realms let's us traverse multiple branches (configured by **.searchBase* property) even if a failure is encountered in any one of them.

Configuring multiple realms:

The rest of the post describes a test setup on how to configure multiple realms using the demo LDAP server that ships with Apache Knox

Creating sample test users

Let's use the users in the example diagram above. For this test we will use the demo LDAP server.

Before starting the demo LDAP server add the following to the {KNOX_HOME}/conf/users.ldif file

Add contractor OU just before the sample users entry

```
# Entry for a sample contractor container
# Please replace with site specific values
dn: ou=contractor,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:organizationalUnit
ou: contractor
```

Add sample users with same uid but different OUs. You can add them after *dn: uid=tom,ou=people,dc=hadoop,dc=apache,dc=org* entry.

```
# entry for sample user jerry
dn: uid=jerry,ou=people,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:person
objectclass:organizationalPerson
objectclass:inetOrgPerson
cn: jerry
sn: jerry
uid: jerry
userPassword:jerry-password
# entry for sample user jerry (contractor)
dn: uid=jerry,ou=contractor,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:person
objectclass:organizationalPerson
objectclass:inetOrgPerson
cn: jerry
sn: jerry
uid: jerry
userPassword:other-jerry-password
```

Now we have same uid (jerry) in two different OUs

Update topology file:

Open the sandbox.xml topology file (or the one that you want to test).

Under the ShiroProvider (Shiro authentication provider) remove or comment out everything between property '*main.ldapRealm*' and property '*main.ldapRealm.contextFactory.authenticationMechanism*' and replace it with the following:

```
                    <param>
                <name>main.ldapRealm</name>
                <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</value>
            </param>
            <param>
                <name>main.ldapContextFactory</name>
                <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
            </param>
            <param>
                <name>main.ldapRealm.contextFactory</name>
                <value>$ldapContextFactory</value>
            </param>
            <param>
                <name>main.ldapRealm.userSearchAttributeName</name>
                <value>uid</value>
            </param>

            <param>
                <name>main.ldapRealm.contextFactory.systemUsername</name>
                <value>uid=guest,ou=people,dc=hadoop,dc=apache,dc=org</value>
            </param>

            <param>
                <name>main.ldapRealm.contextFactory.systemPassword</name>
                <value>guest-password</value>
            </param>
```

```
<param>
    <name>ldapRealm.userObjectClass</name>
    <value>person</value>
</param>

<!-- Search Base for Realm 1 -->
<param>
    <name>main.ldapRealm.searchBase</name>
    <value>ou=people,dc=hadoop,dc=apache,dc=org</value>
</param>

<param>
    <name>main.ldapRealm.contextFactory.url</name>
    <value>ldap://localhost:33389</value>
</param>
<param>
    <name>main.ldapRealm.contextFactory.authenticationMechanism</name>
    <value>simple</value>
</param>

<!-- REALM #2 Start  -->
<param>
    <name>main.ldapRealm2</name>
    <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</value>
</param>
<param>
    <name>main.ldapContextFactory</name>
    <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
</param>
<param>
    <name>main.ldapRealm2.contextFactory</name>
    <value>$ldapContextFactory</value>
</param>

<param>
    <name>main.ldapRealm2.userSearchAttributeName</name>
    <value>uid</value>
</param>

<param>
    <name>main.ldapRealm2.contextFactory.systemUsername</name>
    <value>uid=guest,ou=people,dc=hadoop,dc=apache,dc=org</value>
</param>

<param>
    <name>main.ldapRealm2.contextFactory.systemPassword</name>
    <value>guest-password</value>
</param>

<param>
    <name>ldapRealm.userObjectClass</name>
    <value>person</value>
</param>
<!-- Search Base for Realm 2 -->
<param>
    <name>main.ldapRealm2.searchBase</name>
    <value>ou=contractor,dc=hadoop,dc=apache,dc=org</value>
</param>

<param>
    <name>main.ldapRealm2.contextFactory.url</name>
    <value>ldap://localhost:33389</value>
</param>
<param>
    <name>main.ldapRealm2.contextFactory.authenticationMechanism</name>
    <value>simple</value>
</param>
<!-- REALM #2 End  -->

<!-- Let Knox know about the two different realms -->
```

```
            <param>
              <name>main.securityManager.realms</name>
              <value>$ldapRealm, $ldapRealm2</value>
            </param>
```

In the above example we have defined two realms 'ldapRealm' and 'ldapRealm2'. For more information about the properties look at Apache Knox Advanced LDAP Configuration.

The important properties are described below in the context of multiple realms:

- *main.*.searchBase* - Search base where Knox will start the search
- *main.*.userSearchAttributeName* and */dapRealm.userObjectClass* - LDAP Filter for doing a search, in our case (&(uid=jerry)(objectclass=person))
- *main.securityManager.realms* - Defines the realms to be used for authentication, in this case we use two 'ldapRealm' and 'ldapRealm2'

The advantage of using multiple realms is that if search fails to match user in one realm (using its search base) the search continues in other realm or realms. Although caution is advised in defining the search base to prevent it from being too narrow or restricted.

Test:

Assuming you HDP sandbox, demo LDAP and Apache Knox gateway running on your machine, you can test both the users with the following command and should get a HTTP 200 response

```
curl -i -k -u jerry:jerry-password -X GET 'https://localhost:8443/gateway/sandbox/webhdfs/v1/?op=LISTSTATUS'

curl -i -k -u jerry:other-jerry-password -X GET 'https://localhost:8443/gateway/sandbox/webhdfs/v1/?
op=LISTSTATUS'
```

You could also see login failure when Apache Knox tries to authenticate using a different branch, this of course is expected and shows how Apache Knox traversed to find the right user.

```
# curl -i -k -u jerry:other-jerry-password -X GET 'https://localhost:8443/gateway/sandbox/webhdfs/v1/?
op=LISTSTATUS'
....
....
....
2017-03-01 15:51:45,530 INFO  hadoop.gateway (KnoxLdapRealm.java:getUserDn(724)) - Computed userDn: uid=jerry,
ou=people,dc=hadoop,dc=apache,dc=org using ldapSearch for principal: jerry
2017-03-01 15:51:45,538 INFO  hadoop.gateway (KnoxLdapRealm.java:doGetAuthenticationInfo(203)) - Could not
login: org.apache.shiro.authc.UsernamePasswordToken - jerry, rememberMe=false (0:0:0:0:0:0:0:1)
2017-03-01 15:51:45,538 ERROR hadoop.gateway (KnoxLdapRealm.java:doGetAuthenticationInfo(205)) - Shiro unable
to login: javax.naming.AuthenticationException: [LDAP: error code 49 - INVALID_CREDENTIALS: Bind failed:
ERR_229 Cannot authenticate user uid=jerry,ou=people,dc=hadoop,dc=apache,dc=org]
2017-03-01 15:51:45,541 INFO  hadoop.gateway (KnoxLdapRealm.java:getUserDn(724)) - Computed userDn: uid=jerry,
ou=contractor,dc=hadoop,dc=apache,dc=org using ldapSearch for principal: jerry
```