Security Configuration

Introduction

Lightweight Directory Access Protocol (LDAP) is an application protocol used for accessing a user directory in a network.

The LDAP security components of Moogsoft AIOps are configured in the file called security.conf. You will also need information from your LDAP SME.

From LDAP

LDAP Critical Step	Description
The LDAP URL connection string	ex. "URL": "ldap://LDAPServerName:389"
Identify the User in the LDAP Tree that will have read only rights to search for end users of the tool	userDnLookupUser and userDnLookupPassword
User DN	The Distinguished Name is the unique path to any object in the active directory
"usernameAttribute": "sAMAccountName"	The attribute for the username
"userBaseDn" : "dc=corp,dc=standard,dc=com"	
"userDnLookupUser": "cn=LDAP-Moogsoft_User, ou=Accounts,ou=Users,dc=corp,dc=company, dc=com"	
"userDnLookupPassword": "PASSWORD"	Have the LDAP SME test this password and ensure the password is valid. Most service accounts have disabled the ability to logon which makes it challenging to properly test the password.
groupBaseDn": "OU=Groups,OU=SFG Users, DC=corp,DC=standard,DC=com"	Note: This is can be different than the location of the userDnLookupUser
"memberAttribute": "member"	
"groupNameAttribute": "cn"	

From Moogsoft

What to do	
Edit the Security.conf with all data provided by the LDAP SME as well as your Moogsoft roles	"/usr/share/moogsoft /config/security.conf"
Ldapsearch Utility - It is highly recommended to use this tool as part of your LDAP configuration steps. The tool will enable you to test the existence location of the user being used to search LDAP for Moogsoft end users	
ldapsearch -h LDAPServerName -u CN=LDAP-Moogsoft_User,OU=Accounts,OU=Users,DC=corp,DC=company,DC=com -x -W -b CN=LDAP-Moogsoft_DEV,OU=Groups,OU=Users,DC=corp,DC=company,DC=com	
service moogfarmd stop/start	After any change to the Security.conf file
service apache-tomcat stop/start	After any change to the Security.conf file

Teams Mapping

Team	Description
Role Map ie.."Moogsoft Super User": "Super User"	This is just an example map as many roles as needed to properly configure AIOps
Role Map ie.."Moogsoft Operators": "Operator"	This is just an example map as many roles as needed to properly configure AIOps to map to your groups or teams

Testing and Validation

What to do
The use of the ldapsearch utility will enable you to validate that your configuration settings and user settings are correct
You must engage the LDAP SME for validation of users and configuration settings
Using both ldapsearch and the LDAP SME are the key to a successful LDAP integration
When everything is configured you must stop and restart both moogfarmd and apache-tomcat
The two key logs to check once testing the configuration begins are: catalina.out = /usr/share/apache-tomcat/logs/catalina.out moogfarmd.log /var/log/moogsoft/moogfarmd.log

security.conf

```
$MOOGSOFT_HOME/config/security.conf
```

This file allows different authentication and authorization sources to be defined. Each authentication/authorization source is called a Realm and there are three types:

Realm Type	Description
DB	A moogsoft database
LDAP	An external LDAP server

By default all the Realms within this file are commented out which means the default Moogsoft DB will be used. Edits to this file will require a restart of apache-tomcat for the changes to take effect. If any realms are defined in this file, the default DB Realm will not be used. This leads to the following configuration options:

Option	Description
1	All realms commented out will mean the moogsoft db is used. Only local db users will be able to log into the system
2	One or more LDAP realms (but not DB realm). Only LDAP users will be able to log into the system
3	One or more LDAP realms plus a DB realm (there can only be one db realm). Both LDAP users and local db users will be able to log into the system

If a user is defined in more than one Realm (with or without the same password) they will be authenticated and the login will succeed as long as at least one of the Realms authentication's succeeds.

Depending upon the type of Realm there may be additional configuration items required. The DB Realm does not require additional items but the LDAP one does. If both types are used, the DB Realm must come first in the file. All Realms are defined as follows (the name can be any you choose):

```
"Realm Name" : {
    "realmType": "DB or LDAP",
    ...additional json config ...
    }
```

DB type

An example DB type is given below:

```
{
    #
    # Realm type (DB or LDAP)
    #
    "realmType": "DB"
}
```

LDAP type

An example JSON config with all parameters is given below (each field is described in detail below):

```
{
    "LDAP realm" : {
        "realmType": "LDAP",


        #
        # [LDAP connection section]
        #


        #
        # The url indicates the protocol (ldap, ldaps) as well as the
        # host and port. This field is mandatory.
        #
        "url": "ldap://172.16.124.169:389",


        #
        # If prefefinedUser is true then the user account information
        # must exist in the local DB (as well as the LDAP server).
        #
        # If predefinedUser is false then the LDAP information will
        # be used to create/update the user account.
        #
        # This field is optional (default value is false)
        #
        "predefinedUser": false,


        #
        # [Authentication bind and attribute search section]
        #


        #
        # User DN (Distinguished Name) resolution for bind
        #
        # There are two alternative resolution methods available to determine the
        # user DN. The value of 'resolutionType' controls this and can have one
        # of the following values:
        #
        # "direct" - DN created based on userDnPostfix and usernameAttribute.
        #
        # REQUIRED FIELDS:
        # "usernameAttribute"
        # "userDnPostfix"
        #
        # e.g.
        #
        # For the following config:
        #
        # "resolutionType" : "direct",
        # "direct" : {
        #     "usernameAttribute": "uid",
        #     "userDnPostfix": "ou=People,dc=moogsoft,dc=com"
        # },
        #
        # IF the user typed in a username of 'moo' for the above config then the
        # following DN would be created:
        #
        # "uid=moo,ou=People,dc=moogsoft,dc=com"
        #
        #
        # "lookup" - DN looked up based on userBaseSearchFilter and usernameAttribute.
        #
        # REQUIRED FIELDS:
        # "usernameAttribute"
        # "userBaseDn"
        # "userBaseSearchFilter"
        #
        # OPTIONAL FIELDS:
        # "userDnLookupUser" (if not defined systemUser will be used)
        # "userDnLookupPassword"
        #
        # e.g.
        #
        # For the following config:
        #
        # "resolutionType" : "lookup",
        # "lookup" : {
        #     "usernameAttribute": "sAMAccountName",
        #     "userBaseDn": "ou=People,dc=moogsoft,dc=com",
        #     "userBaseSearchFilter": "(objectclass=people)"
        # },
        #
        # If the user typed in a username of 'moo' for the above config then the
        # following filter would be used (with the userBaseDn):
        #
        # (&(objectclass=people)(sAMAccountName=moo))
        #
        # Note: If lookup matched more than one DN then the authentication would
        # fail.
        #
        # "userDnResolution" : {
        #     "resolutionType" : "direct",
        #     "direct" : {
        #         "usernameAttribute": "uid",
        #         "userDnPostfix": "ou=People,dc=moogsoft,dc=com"
        #     }
        # },
        #
        "userDnResolution" : {
            "resolutionType" : "lookup",
            "lookup" : {
                "usernameAttribute": "sAMAccountName",
                "userBaseDn" : "ou=People,dc=moogsoft,dc=com",
                "userBaseSearchFilter" : "(objectclass=person)",
                "userDnLookupUser": "uid=anon,ou=People,dc=moogsoft,dc=com",
                "userDnLookupPassword": ""
            }
        },


        #
        # The attribute filter will be used in conjunction with the
        # user DN as base to retrieve all user attributes.
        #
        # This field is optional (default '(objectclass=*)')
        #
        "attributeSearchFilter": "(objectclass=*)",


        #
        # The attribute map defines a mapping from user DB fields (in the
        # database) to LDAP attributes. The left hand key is the user DB
        # field name and the right hand side is the LDAP attribute.
        #
```

```
# Note: An LDAP attribute could potentially map to multiple user DB
# fields but a DB field can only map to one LDAP attribute.
#
# Any LDAP attributes not in this list will be ignored.
#
# This field is optional if predefinedUser is true otherwise it is
# mandatory.
#
"attributeMap": {
    "fullname": "cn",
    "email": "mail"
},


#
# [LDAP Group search section]
#


# An optional system user and password will be used to bind
# and search for user group information. If not provided
# then the user DN defined above will be used.
#
# These fields are optional if predefinedUser is true OR the search for
# groups will be done under the user dn depending upon LDAP permissions).
#
"systemUser": "uid=appauth,ou=auth,ou=moogsoft,ou=Applications,dc=moogsoft,dc=com",
"systemPassword": "appauth",


#
# A filter (see below) will be used in conjunction with
# the group base DN to search for LDAP groups.
#
# Group filter ::= "(" <memberAttribute> "=" <UserDN> ")"
#
# The groupBaseDn is optional if predefinedUser is true, otherwise it is
# mandatory. The memberAttribute is optional (default 'member').
#
"groupBaseDn": "ou=groups,ou=moogsoft,ou=Applications,dc=moogsoft,dc=com",
"memberAttribute": "member",


#
# Once the groups have been found using the above DN and filter
# the group name attribute will then be used to get the
# group name. The group name is then used to map the group
# to a role using the role map.
#
# The groupNameAttribute is optional (default 'CN'). The roleMap
# is optional if predefinedUser is true, otherwise it is mandatory.
#
"groupNameAttribute": "cn",
"roleMap": {
    "role-admin": "Super User",
    "role-read": "Operator"
}
        # Add the assignTeams in order to synchronize Teams with
# assignTeams :{
#
#   Set the teamMap as A map from a LDAP group name to AIOps team name
#    teamMap: {
#      "ldap-group-name": "Team name",
#      "yet another group": "Another team"
#    },
#
#    Set the useGroupName to true in order to use the group name as the
#    team name when the group is not in the teamMap nor the roleMap. Default
#    to false.
#    useGroupName : false,
#
```

```
        #   Set the createNewTeams to true in order to create a new team when a
        #   team is unknown in AIOps. Default to false.
        #    createNewTeams: false
        # },

        #
        # # [LDAP Start TLS section]
        # #
        # # Enable Start TLS authentication by specifying the SSL
        # # configuration below.
        # #
        #
        # ########################################################################
        # #                                                                      #
        # # SSL configuration can be used to provide a means of secure           #
        # # communication. SSL can be setup with options to accept SSL           #
        # # connections with or without providing the relevant                   #
        # # certificates and keys.                                               #
        # #                                                                      #
        # # Three modes of SSL are available:                                    #
        # # 1. No SSL      - SSL configuration is not specified                  #
        # #                                                                      #
        # # 2. Express SSL - This is where SSL configuration is specified, but   #
        # #                  empty or only the SSL protocol is set and specific  #
        # #                  certificates do not need to specified.              #
        # #                                                                      #
        # # 3. Custom SSL  - This is where all the SSL configuration and         #
        # #                  certificates needed are specified to enable secure  #
        # #                  and authorised communication.                       #
        # #                                                                      #
        # #                  Note that Client key and certificate are optional.  #
        # #                  If neither of those are specified, then client      #
        # #                  certification verification will not be performaned. #
        # ########################################################################
        #
        # "ssl" :
        # {
        #     #
        #     # Specify the SSL Protocol to use.
        #     # If the configuration is not specified, "TLSv1.2" will be used
        #     # by default.
        #     # JRE 8 supports "TLSv1.2", "TLSv1.1", "TLSv1", "SSLv3"
        #     #
        #     "ssl_protocol" : "TLSv1.2",
        #
        #     #
        #     # The location of the SSL certificate, key files.
        #     #
        #     # Relative pathing can be used, i.e. '.' to mean current directory,
        #     # '../server.pem' or '../../server.pem' etc. If neither relative
        #     # nor absolute (using '/') path is used then $MOOGSOFT_HOME is
        #     # prepended to it.
        #     # i.e. "config/server.pem" becomes "$MOOGSOFT_HOME/config/server.pem"
        #     #
        #
        #     #
        #     # Specify the server certificate.
        #     #
        #     "server_cert_file" : "server.pem",
        #
        #     #
        #     # Enable client authentication by specifying the client certificate
        #     # and key files below.
        #     # The key file has to be in PKCS#8 format.
        #     #
        #     "client_cert_file" : "client.pem",
        #     "client_key_file"  : "client.key"
        # }
        #
    }
}
```

realmType

The type of Realm (authentication/authorisation source). This can be either ‘DB’ or ‘LDAP’.

url

A url as follows:

```
ldap://ip:port
ldaps://ip:port
```

The IP should be the IP address of the LDAP server and the port will typically be 389 (LDAP) and 636 (LDAPS).

By default port 389 is non-SSL, but SSL can be enabled, by upgrading the connection to secure with Start TLS - see Start TLS.
Note that Start TLS is recommended way to get secure connection with SSL, instead of using LDAPS.

If using LDAPS SSL then you will need to import the certificate into JAVA_HOME:

http://docs.oracle.com/javase/tutorial/jndi/ldap/ssl.html

The above link has a section on client requirements:

> Run the following commands as Root

1. Enter the following:

   ```
   # cd JAVA_HOME/lib/security (for JRE ../jre/lib/security)
   ```

2. Check for the cacerts file in:

   ```
   jre/lib/security
   ```

   a. If this file exists, make a copy of it in the same directory and name it: jssecacerts
      This is because if both cacerts and jssecacerts exist, jssecacerts will be used exclusively (so it is best to include the default certificates in jssecacerts by copying it).

3. Run the keytool command:

   ```
   # keytool -import -file server_cert.cer -keystore jssecacerts
   ```

> 1. Java's default cacerts password is "changeit" (for the keytool)
> 2. If you encounter problems after importing the certificate then check:
>   a. That you are importing to the correct JVM (The JVM that is running tomcat).
>   b. Check to see if there is a certificate chain that needs importing as sometimes the whole chain needs to be imported.
> 3. The ‘Start TLS Extension’ is not currently supported (i.e. using ldap:// rather than ldaps:// for a TLS connection):
>   a. http://docs.oracle.com/javase/jndi/tutorial/ldap/ext/starttls.html
>   b. http://www.ietf.org/rfc/rfc2830.txt

predefinedUser

If predefinedUser is true then the user account information must exist in the local DB (as well as the LDAP server). To provision pre-existing users the stored procedure create_remote_user should be used e.g.

```
call create_remote_user ( 'ldapUser','Ldap User', 'All', 'Support', 'Super User', 'Europe/London', -1,
'ldap@moogsoft.com', '@ldap', '555-8765');
```

If predefinedUser is false then the LDAP information will be used to create/update the user account (and the local DB user does not need to exist before logging in).

userDnResolution and resolutionType

A block of config that determines the mechanism used for obtaining the user DN that will be used in the LDAP bind. Within the userDnResolution block resolutionType is mandatory and should have one of two possible values (direct or lookup). Whatever the value there must also be a config block of the same name e.g.

```
“userDnResolution” : {
    “resolutionType” : “direct”,
    “direct” : { … }
}
-or-
“userDnResolution” : {
    “resolutionType” : “lookup”,
    “lookup” : { … }
}
```

direct :

If resolutionType is direct then a user DN will be ‘built’ using the config fields usernameAttribute and userDnPostFix.

Example (“resolutionType” : direct):

```
DN ::= <usernameAttribute> “=” typedUsername ”,” <userDnPostfix>


e.g.
“usernameAttribute” = “uid”
“userDnPostfix” = “ou=People,dc=moogsoft,dc=com”
Typed username = fakeuser


Would lead to the following DN:
uid=fakeuser,ou=People,dc=moogsoft,dc=com
```

lookup :

If resolutionType is lookup then a user DN will be searched for in the LDAP server using the usernameAttribute as a filter and userBaseDn as a base to find the DN. The filter used for the user DN search is a combination (using AND) of userBaseSearchFilter and the usernameAttribute. If the user DN search returned more than one match then the login attempt would fail.

The actual user DN search will either use a specific userDnLookupUser or the systemUser (so at least one of these need to be configured).

Example (“resolutionType” : lookup):

```
User_DN_Filter ::= “(&(” <userBaseSearchFilter> “)(“ <usernameAttribute> “=” typedUsername “))”


e.g.
“userBaseSearchFilter” = “objectclass=Person”
“usernameAttribute” = “sAMAccountName”
Typed username = fakeuser


Would lead to the following user DN search filter:


(&(objectclass=Person)(sAMAccountName=fakeuser)


Note: If no userBaseSearchFilter was specified then the filter would end up as follows:


(sAMAccountName=fakeuser)
```

usernameAttribute

Depending on the value for userDnLookup this key field is used to either build or search for the user DN that is used in the LDAP bind. See userDnResoluti on section for more details.

userDnPostFix

Used in conjunction with the usernameAttribute to build the user DN (Distinguished Name) that is used for authentication. See userDnResolution section for more details.

userBaseDn

When resolutionType is ‘lookup’ this field will be used to determine the base subtree of the LDAP structure of which to do the user DN search.

e.g.

```
 “userBaseDn” : “ou=People,dc=moogsoft,dc=com”
```

userDnLookupUser and userDnLookupPassword

When resolutionType is ‘direct’ the userDnLookupUser and userDnLookupPassword if defined will be used to search for the user DN (based on the entered username). If these fields are not defined then the LDAP module will attempt to use the systemUser and systemPassword instead.

> ⚠ If the userDnLookupPassword is not defined (or is empty) the LDAP module will attempt an unauthenticated user bind (Special configuration would be needed in the LDAP server to allow this)

encryptedUserDnLookupPassword

If you want to encrypt your passwords using the Moog Encryptor, comment out the `userDnLookupUser` and uncomment `encrypteduserDnLookupPassword`.

```
"userDnResolution" : {
            "resolutionType" : "lookup",
            "lookup" : {
                "usernameAttribute": "sAMAccountName",
                "userBaseDn" : "ou=People,dc=moogsoft,dc=com",
                "userBaseSearchFilter" : "(objectclass=person)",
                "userDnLookupUser": "uid=anon,ou=People,dc=moogsoft,dc=com",
                #"userDnLookupPassword": ""#
                #for encrpyting passwords via moog_encrypter comment the userDnLookupPassword, and uncomment
encrypteduserDnLookupPassword
                "encryptedUserDnLookupPassword":""
            }
        },
```

attributeSearchFilter

An LDAP filter that is used for searching for user attributes. This filter will be passed with the user DN as base to find all user attributes. If not specified this will be given the following default value:

```
(objectclass=*)
```

attributeMap

After searching LDAP for user attributes this configuration will be used to set certain user fields (for any mapped LDAP attributes). The format is a follows:

```
 "attributeMap": {
        "db_column_5": "ldap_attribute_1",
        "db_column_2": "ldap_attribute_8",
        "db_column_3": "ldap_attribute_8",
    }
```

The original version of attributeMap had the attributes swapped around so that the LDAP attribute was on the left and the DB column name was on the right. This had serious limitations in that it means an LDAP attribute could only map to a single DB column. Since it is useful to have a single LDAP attribute map to multiple DB columns this was swapped around

systemUser and systemPassword

With the LDAP protocol, the system will send two bind and two search requests. If no system user is configured then the user bind (used for authentication) will be re-used for the LDAP group search also.

groupBaseDn

A DB (Distinguished Name) for the part of the LDAP structure that contains the user groups. This will be used in conjunction with the the memberAttribute t o find any LDAP groups the user belongs to. These groups are then mapped to a local role using the roleMap.

memberAttribute

The member attribute is used to build a group filter to find groups.

```
Filter ::= “(” <memberAttribute> “=” <UserDN> “)”
```

groupNameAttribute

This is the name of the LDAP attribute (for the LDAP group) that identifies the group name. This group name will then be used by the roleMap.

roleMap

Users are linked to Roles via LDAP groups. Users are members of LDAP groups and these groups are used to assign Users Roles. The format is as follows:

```
"roleMap": {
        "role-admin": "Super User",
        "role-other": "Customer"
    }
```

The left hand side of the above mapping is the LDAP group and the right hand side is the Moogsoft role name (as defined in the roles table). The LDAP group names (as defined by the groupNameAttribute e.g. CN) will be mapped to a role using this configuration.

Users who are not members of any of the mapped LDAP groups will not be able to log in.

assignTeams

Users can be assigned to a team via LDAP groups. Users are members of LDAP groups and these groups are used to assign Users Teams. To add this op tional feature add the assignTeams entry to enable it. Please note, if this is enabled, any user assignment to a team done by the UI might get overwritten. This entry should have the following members:

teamMap

A map from the group name in LDAP to the team name in AIOps.

useGroupName

Optional Boolean argument. If set to true, and the group name cannot be found in the teamMap (see above) or the roleMap (see above) it will assign the user to a team with the same name as the LDAP group.

createNewTeams

Optional Boolean argument. If set to true, and AIOps cannot find an existing team with the wanted name (either in the teamMap or when useGroupName is set) it will create a new team. Please note, this team will be created without services, situations filter or alerts filter. This will need to be done using the UI or graze.

Example for the assignTeams format:

```
        assignTeams :{
            teamMap: {
              "ldap-group-name": "Team name",
              "yet_another_group": "Another team"
            },
            useGroupName : true,
            createNewTeams: false
         }
```

Start TLS

ssl

SSL can be enabled for non-SSL connections (by default on port 389), by upgrading the connection to be secure with SSL using Start TLS.

This can be achieved by enabling "*ssl*" configuration section for LDAP Realms. Once enabled, Express or Custom mode can be used.

In Express mode, none of the certificates or keys needs to be specified. Secure connection will be created, even thought the certificate is not validated.

In Custom mode, the provided certificates and keys needs to be valid and match the LDAP Server certificate.

Additional client authentication can be provided with both *client_cert_file* and *cient_key_file* specified, where the *client_key_file* needs to be issued by this same CA as the *server_cert_file*.
Note that client authentication needs to be explicitly enabled in LDAP Server schema.

ssl_protocol

(Optional) Specifies with SSL Protocol to use.

With JRE 8, it can be any of those "TLSv1.2", "TLSv1.1", "TLSv1", "SSLv3".

By default that is "TLSv1.2".

The location of SSL certificate, key files:

Relative pathing can be used, i.e. '.' to mean current directory, '../server.pem' or '../../server.pem' etc.
If neither relative nor absolute (using '/') path is used then $MOOGSOFT_HOME is prepended to it.
i.e. "config/server.pem" becomes "$MOOGSOFT_HOME/config/server.pem"

server_cert_file

Path to the LDAP server certificate file.

client_cert_file

Path to the Client certificate file.

client_key_file

Path to the Client private key file. The provided key needs to be in PKCS#8 format.