Redhat Intergration with Active Directory using SSSD.

There are inherent structural differences between how Windows and Linux handle system users. The user schemas used in Active Directory and standard LDAPv3 directory services also differ significantly. When using an Active Directory identity provider with SSSD to manage system users, it is necessary to reconcile Active Directory-style users to the new SSSD users. There are two ways to achieve it:

- ID mapping in SSSD can create a map between Active Directory security IDs (SIDs) and the generated UIDs on Linux. ID mapping is the simplest option for most environments because it requires no additional packages or configuration on Active Directory.
- Unix services can manage POSIX attributes on Windows user and group entries. This requires more configuration and information within the Active Directory environment, but it provides more administrative control over the specific UID/GID values and other POSIX attributes.

Active Directory can replicate user entries and attributes from its local directory into a global catalog, which makes the information available to other domains within the forest. Performance-wise, the global catalog replication is the recommended way for SSSD to get information about users and groups, so that SSSD has access to all user data for all domains within the topology. As a result, SSSD can be used by applications which need to query the Active Directory global catalog for user or group information.

Before we start, here are few of the links which are helpful.

Blog Tech Uniqe

Access Redhat Documentation

Access Redhat Documentation

Cloudera Documentation

Intro from Redhat Documentation

Background about the setup.

We have our setup as below.

1. Two Active Directory servers. `LAB` and `ANOTHERLABLAB`
2. Two Edge Nodes runnning RHEL 6.6, which can directly communicate with both AD servers.
3. Two Slave Nodes are running behind a firewall, which can only communicate to `EDGE` nodes.

We have to configure Slave to send the traffic to EDGE nodes which will forward the traffic to AD servers.

Preparation for the setup.

`[Interface Forwarding]` from `eth1` to `eth0` on `EDGE` node.

Adding `route` to all the `slaves` which reside on a private network to communicate with `External Server` directly using an `EDGE` node using `Interface Forwarding`.

NOTE : Below testing was done on RHEL 6.6

What we are trying to do.

1. All the slave nodes will send their data to Edge nodes on a private interface.
2. Edge Node will take the data arriving on the `private` interface and forward it over a external interface.

NOTE: below I have used `slave`s for all the nodes which are communicating with `EDGE`, in the this case making `EDGE` as the `master` which acts like a `router`.

Slave `ifconfig`

Slaves will only run on Private network.

1. `192.168.0.8` aka `eth0` Private Interface.

Here is the `ifconfig`.

```
[root@slave-node ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr
          inet addr:192.168.0.8  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::21d:d8ff:feb7:1efe/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:131581 errors:0 dropped:0 overruns:0 frame:0
          TX packets:148636 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11583580 (11.0 MiB)  TX bytes:35866144 (34.2 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:245626 errors:0 dropped:0 overruns:0 frame:0
          TX packets:245626 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:286415155 (273.1 MiB)  TX bytes:286415155 (273.1 MiB)
```

Edge Node `ifconfig`

1. `172.14.14.214` aka `eth0` External Interface
2. `192.168.0.11` aka `eth1` Private Interface.

Here is the `ifconfig`.

```
[root@edge-node ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr
          inet addr:172.14.14.214  Bcast:172.14.14.255  Mask:255.255.255.0
          inet6 addr: fe80::21d:d8ff:feb7:1f7b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:908442 errors:0 dropped:0 overruns:0 frame:0
          TX packets:235173 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:77363514 (73.7 MiB)  TX bytes:33167098 (31.6 MiB)
```

```
eth1      Link encap:Ethernet  HWaddr
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::21d:d8ff:feb7:1f7a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:210510 errors:0 dropped:0 overruns:0 frame:0
          TX packets:177170 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:61583138 (58.7 MiB)  TX bytes:16125613 (15.3 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:13799253 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13799253 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:27734863794 (25.8 GiB)  TX bytes:27734863794 (25.8 GiB)

[root@edge-node ~]#
```

Configuration.

1. Create `FORWARD`er on the Edge node.
2. Create `route` on all the `slave`.
3. Update `/etc/hosts` on `slave` nodes.

1. Create `FORWARD`er on the `Edge` node.

1. If you haven't already enabled forwarding in the kernel, do so.
2. Open `/etc/sysctl.conf` and uncomment `net.ipv4.ip_forward = 1`
3. Then execute `$ sudo sysctl -p`
4. Add the following rules to `iptables`

Commands.

```
[root@edge-node ~]# iptables -t nat -A POSTROUTING --out-interface eth0 -j MASQUERADE
[root@edge-node ~]# iptables -A FORWARD --in-interface eth1 -j ACCEPT
```

2. Create `route` on all the `slave`.

Here is the command to add the route in `slaves`.

```
[root@datanode ~]# route add -net 172.0.0.0 netmask 255.0.0.0 gw 192.168.0.11 eth0
```

We are tell all the traffic trying to go to `172.x.x.x` will have to use `192.168.0.11` as the gateway. Which is the `Private` Interface on the `Edge` Node.

3. Update `/etc/resolve.conf` on `slave` nodes.

And then we update the `/etc/resolve.conf` file with the direct `IP` of External Server `172.14.14.174`, as `slave` node now should be able to communicate to the External Server.

```
; generated by /sbin/dhclient-script
nameserver 172.14.14.174
```

Testing if `ping` works.

```
[root@hadooptest-slave ~]# ping hadooptestdc.lab.com
PING hadooptestdc.lab.com (172.14.14.174) 56(84) bytes of data.
64 bytes from 172.14.14.174: icmp_seq=1 ttl=127 time=0.866 ms
64 bytes from 172.14.14.174: icmp_seq=2 ttl=127 time=1.09 ms
64 bytes from 172.14.14.174: icmp_seq=3 ttl=127 time=1.12 ms
64 bytes from 172.14.14.174: icmp_seq=4 ttl=127 time=0.933 ms
^C
--- hadooptestdc.lab.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7042ms
rtt min/avg/max/mdev = 0.866/1.004/1.122/0.112 ms
[root@hadooptest-slave ~]#
```

Preparation for SSSD.

Prerequisite installations.

```
yum install sssd sssd-client krb5-workstation samba openldap-clients open-ssl authconfig
```

we need to `resolve.conf` so that we can contact the AD server using a FQDN.

```
; generated by /sbin/dhclient-script
nameserver 172.14.14.174 ;Domain controller IP also acts like DNS.
```

FQDN for out AD server is.

```
hadooptestdc.lab.com
```

Testing if `ping` works.

```
[root@hadooptest-slave ~]# ping hadooptestdc.lab.com
PING hadooptestdc.lab.com (172.14.14.174) 56(84) bytes of data.
64 bytes from 172.14.14.174: icmp_seq=1 ttl=127 time=0.866 ms
64 bytes from 172.14.14.174: icmp_seq=2 ttl=127 time=1.09 ms
64 bytes from 172.14.14.174: icmp_seq=3 ttl=127 time=1.12 ms
64 bytes from 172.14.14.174: icmp_seq=4 ttl=127 time=0.933 ms
^C
--- hadooptestdc.lab.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7042ms
rtt min/avg/max/mdev = 0.866/1.004/1.122/0.112 ms
[root@hadooptest-slave ~]#
```

Setting krb5 configuration.

Setting up the krb setting to communicate with AD using Kerberos.

```
[libdefaults]
default_realm = LAB.COM
dns_lookup_kdc = false
dns_lookup_realm = false
ticket_lifetime = 86400
renew_lifetime = 604800
forwardable = true
default_tgs_enctypes = rc4-hmac
default_tkt_enctypes = rc4-hmac
permitted_enctypes = rc4-hmac
udp_preference_limit = 1
[realms]
LAB.COM = {
kdc = hadooptestdc.lab.com
admin_server = hadooptestdc.lab.com
}

ANOTHERLABLAB.COM = {
kdc = ersdcserver.anotherlab.com
admin_server = ersdcserver.anotherlab.com
}

[domain_realm]
anotherlab.com = ANOTHERLABLAB.COM
.anotherlab.com = .ANOTHERLABLAB.COM
lab.com = LAB.COM
.lab.com = .LAB.COM


[logging]
kdc = FILE:/var/krb5/log/krb5kdc.log
admin_server = FILE:/var/krb5/log/kadmin.log
default = FILE:/var/krb5/log/krb5lib.log
```

Testing krb5 setup.

Once we have the configuration we will use `kinit` to test.

```
[root@hadooptest-slave ~]# kinit zxa13ahmd@LAB.COM
Password for zxa13ahmd@LAB.COM:
[root@hadooptest-slave ~]# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: zxa13ahmd@LAB.COM

Valid starting     Expires            Service principal
09/12/15 08:37:56  09/12/15 18:38:03  krbtgt/LAB.COM@LAB.COM
        renew until 09/19/15 08:37:56
[root@hadooptest-slave ~]# klist -e
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: zxa13ahmd@LAB.COM
```

```
Valid starting     Expires            Service principal
09/12/15 08:37:56  09/12/15 18:38:03  krbtgt/LAB.COM@LAB.COM
        renew until 09/19/15 08:37:56, Etype (skey, tkt): arcfour-hmac, aes256-cts-hmac-sha1-96
```

Now we are able to connect to ldap and get the tgt as well. so we are ready for the next steps.

Testing `ldapsearch` from the Linux server.

This step is to make sure that our active directory is accessible. And we are able to search users and groups from Linux nodes. Goto linux machine and execute the below command.

```
ldapsearch -v -x -H ldap://hadooptestdc.lab.com/ -D "cn=zxa13ahmd,cn=Users,dc=lab,dc=com" \
                                                  -W -b "cn=cmadadmin,ou=cmlab,dc=lab,dc=com"
```

Here are some more details about the options above.

More details here : `http://linuxcommand.org/man_pages/ldapsearch1.html`

```
-v      Run in verbose mode, with many diagnostics written to standard output.
-x      Use simple authentication instead of SASL.
-H ldapuri
        Specify URI(s) referring to the ldap server(s).
-D binddn
        Use the Distinguished Name binddn to bind to the LDAP directory.
-W      Prompt for simple authentication.  This is used instead of specifying the password on the commar
-b searchbase
        Use searchbase as the starting point for the search  instead  of the default.
```

Above we are trying to search for information about `cmadadmin` using the user `zxa13ahmd`. When you execute the command above we need to enter the password for `zxa13ahmd`.

Creating SSSD Configuration.

Finally we are ready to configure SSSD. Below are the SSSD configuration to connect to `lab.com`.

If we want to connect to multiple AD servers, then we need to add multiple `[domain/anotherlab.com]` in the configuration.

```
[sssd]
config_file_version = 2
debug_level = 0
domains = lab.com, anotherlab.com
services = nss, pam

[nss]
filter_groups = root
filter_users = root
reconnection_retries = 3
entry_cache_timeout = 3
entry_cache_nowait_percentage = 75
debug_level = 8
account_cache_expiration = 1
```

```
[pam]
reconnection_retries = 3

[domain/lab.com]
debug_level = 8
id_provider = ldap
auth_provider = ldap
chpass_provider = krb5
access_provider = simple
cache_credentials = false
min_id = 1000
ad_server = hadooptestdc.lab.com
ldap_uri = ldap://hadooptestdc.lab.com:389
ldap_schema = ad
krb5_realm = LAB.COM
ldap_id_mapping = true
cache_credentials = false
entry_cache_timeout = 3
ldap_referrals = false
ldap_default_bind_dn = CN=zxa13ahmd,CN=Users,DC=lab,DC=com
ldap_default_authtok_type = password
ldap_default_authtok = Welcome123
fallback_homedir = /home/%u
ldap_user_home_directory = unixHomeDirectory

#####################################################
# Update below with another AD server as required #
#####################################################

[domain/anotherlab.com]
debug_level = 8
id_provider = ldap
auth_provider = ldap
chpass_provider = krb5
access_provider = simple
cache_credentials = false
min_id = 1000
ad_server = ATL1ANOTHERLABLABDC1.anotherlab.com
ldap_uri = ldap://ATL1ANOTHERLABLABDC1.anotherlab.com/:389
ldap_schema = ad
krb5_realm = ANOTHERLABLAB.COM
ldap_id_mapping = true
cache_credentials = false
entry_cache_timeout = 3
ldap_referrals = false
ldap_default_bind_dn = CN=anotherlabuser2,CN=Users,DC=anotherlab,DC=com
ldap_default_authtok_type = password
ldap_default_authtok = zubair@123
fallback_homedir = /home/%u
ldap_user_home_directory = unixHomeDirectory
```

Install `oddjob-mkhomedir` to auto create the directory whenever a user logs in.

```
yum install oddjob-mkhomedir
```

Enable `sssd`, `localauth` and update the configuration.

```
authconfig --enablesssd --enablesssdauth --enablelocauthorize --update
```

NOTE : Check the `sssd.conf` again, sometimes `authconfig` will insert the `default` domain. You can remove it and make the `sssd.conf` file similar to what we have above.

Start `sssd` services.

```
service sssd start
service oddjobd start
```

Testing our setup.

Checking our user, who is present in the Active Directory.

```
[root@hadooptest-slave ~]# id zxa13ahmd
uid=62601149(zxa13ahmd) gid=62600513(Domain Users) groups=62600513(Domain Users),
62601134(supergroup),62601133(hdfs)
[root@hadooptest-slave ~]# su zxa13ahmd
[zxa13ahmd@hadooptest-slave root]$ cd ~
[zxa13ahmd@hadooptest-slave ~]$ pwd
/home/zxa13ahmd
```

Next we try to login from remote.

```
[zxa13ahmd@hadooptest-slave ~]$ exit
exit
[root@hadooptest-slave ~]# ssh zxa13ahmd@192.168.0.9
zxa13ahmd@192.168.0.9's password:
Last login: Sat Sep 12 07:46:15 2015 from hadooptest-slave.lab.com
[zxa13ahmd@hadooptest-slave ~]$ pwd
/home/zxa13ahmd
[zxa13ahmd@hadooptest-slave ~]$ id
uid=62601149(zxa13ahmd) gid=62600513(Domain Users) groups=62600513(Domain Users),
62601133(hdfs),62601134(supergroup)
[zxa13ahmd@hadooptest-slave ~]$
```

We are able to and the `/home/zxa13ahmd` is autocreated when the user logged-in.

Now checking users for ANOTHERLAB Domain.

```
[root@hadooptest-slave ~]# id anotherlabuser2
uid=1916401111(anotherlabuser2) gid=1916400513 groups=1916400513,1916401114(supergroup-test),
1916401113(hadoop-test),1916401112,1916401112
[root@hadooptest-slave ~]# id anotherlabuser2
uid=1916401111(anotherlabuser2) gid=1916400513 groups=1916400513,1916401114(supergroup-test),
1916401113(hadoop-test),1916401112,1916401112
[root@hadooptest-slave ~]# su anotherlabuser2
sh-4.1$ pwd
/root
sh-4.1$ cd ~
sh-4.1$ pwd
/home/anotherlabuser2
sh-4.1$ id
```

```
uid=1916401111(anotherlabuser2) gid=1916400513 groups=1916400513,1916401112,
1916401113(hadoop-test),1916401114(supergroup-test)
sh-4.1$ exit
exit
```

We are done.