Exercise 05.03: Authentication and Authorization

In this exercise, you will accomplish the following:

- Enable authentication and authorization
- Create additional users
- Assign roles to users

Full Cassandra security is an in-depth topic. The purpose of this exercise is to become familiar with some of the main security concepts and concerns. This exercise also serves as foundational to more comprehensive Cassandra security. As such, the procedures used here should *not* be considered best practices.

Background

The starting point for this exercise is to have a two-node cluster on the nodes named DSE-node1 and DSE-node3. Make certain the cluster is up and running. Having enabled JMX authentication in the previous exercise, user credentials are now required in order to log in and run the `nodetool` command:

```
nodetool -u ubuntu -pw ubuntu status
```

Confirm the Cassandra process is running on both nodes; node1 and node3.

Definitions

- *Authentication* means having to supply credentials (i.e., a username and password) when performing operations, such as with `cqlsh`.
- *Authorization* means allowing specific users, or groups of users (known as roles), to be able to perform certain types of operations (SELECT, ALTER, INSERT) on specified resources (such as tables).

Step 1: Configuring Authentication

Enabling authentication and authorization within DSE is relatively simple.

1. On DSE-node1, enable authentication and authorization by modifying the file *cassandra.yaml*. As a reminder, this file is located in the */home/ubuntu/dse/conf* directory.

2. Modify the following line to read as follows:

```
authenticator: PasswordAuthenticator
```

Note that the default authenticator option is *AllowAllAuthenticator*.

3. Restart the node1 server when done editing the file.

```
ps -ef | grep cassandra
kill <cassandra_pid>
/home/ubuntu/dse/bin/cassandra
```

4. Log into the DSE-node1 server when Cassandra is operational. Note that node1 is still using the default cassandra user and password.

```
cqlsh -u cassandra -p cassandra
```

5. To prevent security breaches, replace the default superuser, *cassandra*, with different super-user with a different name and secure password:

```
CREATE ROLE <new_superuser> WITH PASSWORD = '<new_secure_password>' AND
SUPERUSER=true AND LOGIN=true;
```

CAUTION: Be aware that the default user *cassandra* reads with a consistency level of `QUORUM` by default, whereas any other superuser reads with a consistency level of `LOCAL_ONE`.

6. Exit out of `cqlsh` and log in as the newly created superuser on node1:

```
cassandra@cqlsh> exit
ubuntu@DSE-node1:~/dse/bin$ cqlsh -u admin -p Letmein12345
Connected to KillrVideoCluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.3.5113 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.
admin@cqlsh>
```

7. It is possible to *neutralize* the default account by changing the password to something long and unable to be guessed, and also alter the user's status to NOSUPERUSER:

```
admin@cqlsh> ALTER ROLE cassandra WITH
PASSWORD='ALongAndUnintelligiblePassword' AND SUPERUSER=false;
```

8. Delete the account by logging into `cqlsh` using the new superuser account created in the previous step and then running the command DROP ROLE.

```
DROP ROLE cassandra;
```

9. Attempt to log back into node1 using the old cassandra username and password. Note the following error should appear:

```
cqlsh -u cassandra -p cassandra
Connection error: ('Unable to connect to any servers', {'127.0.0.1':
AuthenticationFailed('Failed to authenticate to 127.0.0.1: Error from
server: code=0100 [Bad credentials] message="Provided username
cassandra and/or password are incorrect"',)})
```

Step 2: Configuring Authorization

Having created a new role, authorize this role to access database objects. Understand that fetching authorization for a role can be a costly operation.

1. Change the option *authorizer* in the *cassandra.yaml* file from the default of 'AllowAllAuthorizer' to the following:

```
authorizer: CassandraAuthorizer
```

NOTE: Unlike the 'PasswordAuthenticator' option for authentication, authorization relies upon user roles and *not* password access, hence the difference in syntax.

2. To decrease the burden on the system, adjust the validity period for role caching by editing the 'roles_validity_in_ms' option in the *cassandra.yaml* file (default 2000 milliseconds). To disable, set this option to 0:

```
roles_validity_in_ms: 0
```

NOTE: This setting is automatically disabled when the authenticator is set to 'AllowAllAuthenticator.'

3. Configure the refresh interval for role caches by setting the 'roles_update_interval_in_ms' option in the *cassandra.yaml* file (default 2000 ms). If the previous variable is set to 0, then this must also be set to 0:

```
roles_update_interval_in_ms: 0
```

NOTE: If the previous option 'roles_validity_in_ms' is non-zero, this setting must be enabled and set to a value. The credentials are cached in their encrypted form.

4. Similar to authentication, fetching authentication for credentials can also be a costly operation. To decrease the burden on the system, adjust the validity period for credential caching with the 'credentials_update_interval_in_ms' option in the *cassandra.yaml* file (default 2000 ms). Set this value to 0 to disable credentials caching:

```
credentials_validity_in_ms: 0
```

This setting is automatically disabled when the authenticator is set to 'AllowAllAuthenticator.'

5. To set the refresh interval for credentials caches, use the 'credentials_update_interval_in_ms' option (default 2000 ms). If the option 'credentials_validity_in_ms' is set to a non-zero value, this option must be enabled and set:

```
credentials_update_interval_in_ms: 2000
```

6. To disable configuration of authentication and authorization caches (credentials, roles, and permissions) via JMX, uncomment the following line in the *jvm.options* file:

```
-Dcassandra.disable_auth_caches_remote_configuration=true
```

7. To enable the new settings, restart the database.

```
ps -ef | grep cassandra
kill <cassandra_pid>
/home/ubuntu/dse/bin/cassandra
```

Step 3: Enable Authentication and Authorization on Additional Nodes

1. Enable the same authentication and authorization changes to the *cassandra.yaml* and *jvm.options* files for DSE-node3. Walk through Steps 1 and 2 for DSE-node3.
2. Note that the new username and password should have propagated to node3. When logging in using `cqlsh`, use the new admin username. Also, the username 'cassandra' should also have been dropped due to replication.
3. When done, restart the cassandra service on DSE-node3:

```
ps -ef | grep cassandra
kill <cassandra_pid>
/home/ubuntu/DSE/bin/cassandra
```

4. Wait for the node to come up. Once it is running, log into DSE-node3 using the same username and password configured in the previous exercise.

```
nodetool -u ubuntu -pw ubuntu status
```

5. Restart the cassandra process on DSE-node1:

```
ps -ef | grep cassandra
kill <cassandra_pid>
/home/ubuntu/dse/bin/cassandra
```

6. Wait for this node and Cassandra process to be fully running:

```
nodetool -u ubuntu -pw ubuntu status
```

7. Try and launch `cqlsh`.

```
cqlsh `hostname -i`
```

8. Notice that `cqlsh` does not grant access to just any user. It now requires users to supply credentials. Take note of the error message.

```
Connection error: ('Unable to connect to any servers',
{'172.31.17.203': AuthenticationFailed('Remote end requires
authentication',)})
```

9. Use the new `cqlsh` superuser account 'admin' with a password of `Letmein12345`. Login using this account:

```
cqlsh `hostname -i` -u admin -p Letmein12345
Connected to KillrVideoCluster at 172.31.17.203:9042.
[cqlsh 5.0.1 | Cassandra 3.11.3.5113 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.
ubuntu@cqlsh>
```

10. Cassandra stores the authentication and authorization information in the 'system_auth' keyspace. There is some exposure at present in that this keyspace only has a replication factor of one (1). Were we to lose a single node, we would not be able to log into the database. The first thing to do is change the replication factor on this keyspace. While still in the `cqlsh` interface, run the following commands:

```
ALTER KEYSPACE "system_auth" WITH REPLICATION = {'class' :
'NetworkTopologyStrategy', 'dc1' : 2};
```

11. Run repair immediately on the DSE-node3 to confirm the node replicated the authentication and authorization data:

```
nodetool –u admin –pw Letmein12345 repair
```

12. Confirm the changes are in place by running the following command:

```
admin@cqlsh> DESCRIBE KEYSPACE system_auth;
CREATE KEYSPACE system_auth WITH replication = {'class':
'NetworkTopologyStrategy', 'dc1': '2'} AND durable_writes = true;
```

Step 4: Secure the Cluster

1. Secure the cluster by changing the default password for the ubuntu account. On the DSE-node1 instance and in `cqlsh`, execute the following command:

```
ALTER USER ubuntu WITH PASSWORD 'new_password';
```

2. If an error message pops up showing that user does not exist, create the role using the following command:

```
CREATE ROLE ubuntu WITH PASSWORD='new_password' and SUPERUSER=true AND
LOGIN=true;
```

The variable 'new_password' is the new password assigned to that particular user.

3. Attempt to log into DSE-node3 via `cqlsh` using the old ubuntu user password. Has the ubuntu user password also replicated to this node?

Step 5: Adding Users and Roles

Next, create a new user. It is possible to create users either *with* or *without* super-user privileges. Run all the following commands on DSE-node1.

1. Create a user named 'wilma' as a super-user, and the user 'fred,' who is not a super-user. In `cqlsh` users and roles are technically the same thing. Consider a user to be a role associated with only one person.
2. On DSE-node1 and in `cqlsh`, execute the following command:

```
CREATE ROLE wilma WITH PASSWORD = 'Pebbles123' AND SUPERUSER=true AND
LOGIN=true;
CREATE ROLE fred WITH PASSWORD = 'YabbaDabbaDoo123' AND LOGIN=true;
```

3. List the roles:

```
LIST ROLES;
```

4. You should see something similar to the following: Note that both users, 'wilma' and 'fred,' have been added. Take note of their superuser status. Did you note this when creating their roles?

```
role    | super | login | options
--------+-------+-------+---------
  admin |  True |  True |        {}
   fred | False |  True |        {}
 ubuntu |  True |  True |        {}
  wilma |  True |  True |        {}
```

```
(4 rows)
admin@cqlsh>
```

5. Remove the user 'wilma' with the following command:

```
DROP ROLE wilma;
LIST ROLES;
```

Step 6: Managing Privileges

1. It is also possible to grant privileges directly to a user:

```
GRANT SELECT ON killr_video.user_by_email TO fred;
```

2. This command permits the user 'fred' to query the 'user_by_email' table. Note that 'fred' can query 'user_by_email' but cannot insert any data into that table.
3. It is also possible to grant privileges indirectly. Do this by creating a role, granting privileges to that role, and then assigning that role to a user:

```
CREATE ROLE modifier WITH PASSWORD = 'VerySecurePassword123';
GRANT MODIFY ON killr_video.user_by_email TO modifier;
GRANT modifier TO fred;
LIST ROLES;
```

4. The user 'fred' can now modify (i.e., INSERT, DELETE, UPDATE, TRUNCATE) the table.
5. It is also possible to revoke privileges per the following example:

```
REVOKE SELECT ON killr_video.user_by_email FROM fred;
```

Reference

The following table is a reference list of the various privileges:

Permission	CQL Statements
SELECT	SELECT
ALTER	ALTER KEYSPACE, ALTER TABLE, CREATE INDEX, DROP INDEX
AUTHORIZE	GRANT, REVOKE
CREATE	CREATE KEYSPACE, CREATE TABLE
DROP	DROP KEYSPACE, DROP TABLE
MODIFY	INSERT, DELETE, UPDATE, TRUNCATE

END OF EXERCISE