Implementation of Network Policy - SNOWFLAKE

L'oreal - IP WHITELISTING

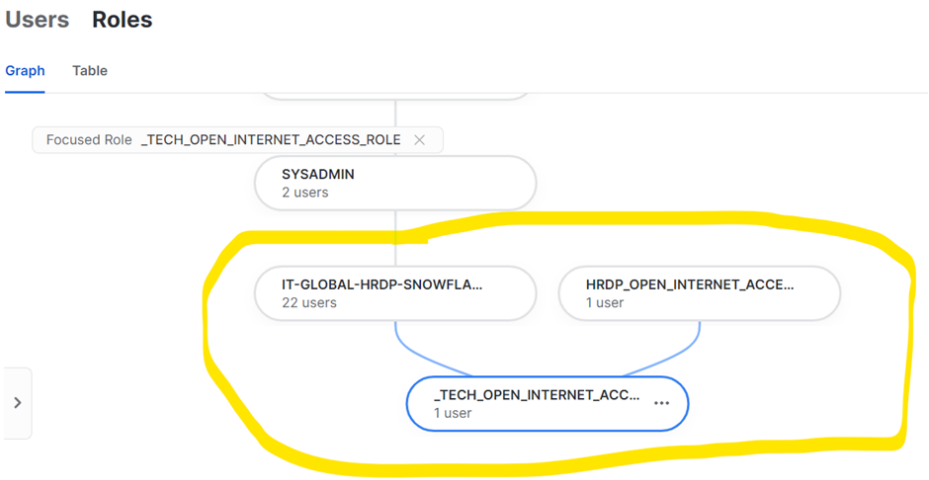
There are five applications need to be whitelisted,

1. Power BI
2. DBT Cloud
3. GCP
4. Open Internet
5. AAD Provisioning Security Integration

For all these applications individual network policy except aad provisioning has been created and listed below,

1. **ACCOUNT_NETWORK_POLICY**
2. **HRDP_DBT_NETWORK_POLICY**
3. **HRDP_AIRFLOW_<env>_NETWORK_POLICY**
4. Through the role **(_TECH_OPEN_INTERNET_ACCESS_ROLE)** and one hierarchy above to this role. The users present till this hierarchy will have open internet access using the policy **HRDP_OPEN_NETWORK_POLICY**.

As per the below highlighted, 23 users will have access for open internet**.**



Apart from POWERBI IP's all of the remaining are STATIC Ips'.

Step 1:

For Microsoft Power BI Ip's it will get refreshed twice in a month. These IP's are captured and placed in a GCP bucket under this path using the python code.

Note: We know only this file is getting refreshed on Monday but not the exact time interval. So we have scheduled the stored procedure to run every 3 hours on Monday.

Step 2:

One time Process in snowflake.

Created the database **HRDP_ADM_DV_DB**

Created the schema **IP_WL**

Created the file format **HRDP_PBI_IP_FF**

Created the stage to get authenticated with GCP bucket URL **HRDP_PBI_IP_STAGE**

Created the stored procedure (**extract_azure_ips_sp**) to pull the Json file from GCP bucket and store the json file in the table AZURE_TAGS_IP_RANGES.

```
create or replace procedure extract_azure_ips_sp

(FILENAME STRING)

returns string not null

language javascript

    Execute as caller as

$$

   var sql_to_clean_json_file = `CREATE OR REPLACE TABLE AZUREDB(src variant) as select value as src from table(flatten(select parse_json($1): values from @HRDP_PBI_IP_STAGE/` + FILENAME +` (file_format => 'HRDP_PBI_IP_FF')));` ;

   var statement1 = snowflake.createStatement( {sqlText:sql_to_clean_json_file} );

   var result_set1 = statement1.execute();

     // Uncomment the following line to see the generated statement.

   //return sql_to_clean_json_file;

   // Step 2. Transform the result into a table. Extract the IP address

   // array into a string of comma-separated IP addresses for use in a Network

// Policy.

   var sql_to_extract_ips_and_service_tags = "";

   var sql_to_extract_ips_and_service_tags = `create or replace table AZURE_TAGS_IP_RANGES

    (SERVICE_TAGS string,REGIONS string,IP_Prefixes String)

     as select src:id::string as

          SERVICE_TAGS,src:region::string as REGIONS,concat

                                        ('\\",

                array_to_string(src:properties:addressPrefixes,

                '\\',\\"), '\\")

               as IP_Prefixes from  AZUREDB;`;

   var statement2 = snowflake.createStatement( {sqlText:sql_to_extract_ips_and_service_tags} );

   var result_set2 = statement2.execute();

   return "success";

$$;


call extract_azure_ips_sp('power_bi_ip_list.json');
```

Then create the table IPS_OTHER_APPS which will have the other application IP’s such as DBT IP’s as of now.

```
insert into IPS_OTHER_APPS (app_name,ip)
values  ('DBT','52.45.144.63')
,('DBT','54.81.134.249')
,('DBT','52.22.161.231')
,('DBT','52.3.77.232')
,('DBT','3.214.191.130')
,('DBT','34.233.79.135');
```

Final step is to create the account level network policy (ACCOUNT_NETWORK_POLICY) which will accomodate the PBI IP’s and the IP’s from other application.

```
create or replace procedure account_level_net_pol -- to get all the pbi ip's and union all the ip's from IPS_OTHER_APPS table
(SERVICETAG varchar
,DRIVING_TABLE varchar)
returns string not null
language javascript
    Execute as caller as
$$
   // Checking for no null IPS for PBI
   var count_PBI=`select count(*) from AZURE_TAGS_IP_RANGES where SERVICE_TAGS='` + SERVICETAG + `'`;
   var statement_count_1 = snowflake.createStatement( {sqlText: count_PBI} ).execute();
      while (statement_count_1.next()) {
         var count_numb1 = statement_count_1.getColumnValue(1);
         try {
        if ( count_numb1 === 0 ) {
            throw `No IPs in table for ` + SERVICETAG + `;`;
        } else {
        }
    } catch (err) {
        throw err;
      }}
   // Checking for no null IPS for DBT
    var count_DBT=`select count(*) from ` + DRIVING_TABLE +`;`;
   var statement_count_2 = snowflake.createStatement( {sqlText: count_DBT} ).execute();
   while (statement_count_2.next()) {
```

```
        var count_numb2 = statement_count_2.getColumnValue(1);

        try {

       if ( count_numb2 === 0 ) {

           throw `No IPs in table ` + DBT_TABLE +` ;`

       } else {

       }

    } catch (err) {

       throw err;

     }}

   // Ignore IPV6 and Get the list of IPv4 to be inserted into the service tag Network Policy.

   // Plus Merge IPs from DBT

   var sql_to_get_list_of_service_ips = "";

 var sql_to_get_list_of_service_ips = `select LISTAGG(IPS, ',') as IPS from (select split_part(IP_Prefixes, ',\\'2603:',1) as IPS from
AZURE_TAGS_IP_RANGES where SERVICE_TAGS='` + SERVICETAG + `' union select '\\"||LISTAGG(trim(IP),'\\',\\")||'\\" as IPS from ` +
DRIVING_TABLE +` );`;

   var statement1 = snowflake.createStatement( {sqlText: sql_to_get_list_of_service_ips} );

   var result_set1 = statement1.execute();

   var get_the_first_row = result_set1.next();

   var ips_column_name = "IPS"

   var quota_separated_list_of_ips = result_set1.getColumnValue(ips_column_name);

   // Uncomment the following line to see the generated statement:

   //return quota_separated_list_of_ips;

   // Create the Network Policy for that service tag without activating it.

   // Only activate the Network Policy after verifying it!

   var sql_to_create_network_policy = "";

   var sql_to_create_network_policy = `create network policy ACCOUNT_NETWORK_POLICY ALLOWED_IP_LIST =

                              (` + quota_separated_list_of_ips + `);`

   var statement2 = snowflake.createStatement( {sqlText:sql_to_create_network_policy});

    try {

       statement2.execute();

       return sql_to_create_network_policy;

       }

       catch (err)

       {

       var sql_to_alter_network_policy = `alter network policy ACCOUNT_NETWORK_POLICY set ALLOWED_IP_LIST =

                              (` + quota_separated_list_of_ips + `);`

       var statement3 = snowflake.createStatement( {sqlText:sql_to_alter_network_policy});

       statement3.execute();

       return sql_to_alter_network_policy;

       }

       $$;
```

```
call account_level_net_pol('PowerBI','IPS_OTHER_APPS');
```

To schedule this procedure using the below statements,

```
CREATE OR REPLACE TASK HRDP_ACCOUNT_NETWORK_POLICY_TASK
  WAREHOUSE = HRDP_DBT_BASE_WH
  SCHEDULE = 'USING CRON 0 0/3 * * MON UTC'
  AS
    call account_level_net_pol('PowerBI','IPS_OTHER_APPS');
```

Below is the procedure for open network access

```
CREATE OR REPLACE PROCEDURE OPEN_IP_ADMIN_SP(
    DRIVING_ROLE varchar,
    NETWORK_POLICY varchar,
    DRIVING_TABLE varchar
  )
  RETURNS varchar
  LANGUAGE JAVASCRIPT
  COMMENT = 'SP used to whitelist the users associated to a set of roles'
  EXECUTE AS CALLER
  AS
$$
//Creating Variables
let v_driving_role = DRIVING_ROLE;
let v_network_policy = NETWORK_POLICY ;
let v_driving_table = DRIVING_TABLE ;
//Sql command to join Account usage user to role, to list of users
let v_return_users_sql = `with hierarchy as
          (select
              name as parent_id ,
              name       as child_id ,
               1 level,
              array_construct(child_id) path,
              name as top_role
```

```
        from   snowflake.account_usage.roles  r
        where deleted_on is null
        union all
        select
              pc.grantee_name ,
              pc.name ,
              h.level+1 level ,
              array_cat(h.path, array_construct(pc.name)) path,
              h.top_role
        from snowflake.account_usage.grants_to_roles pc
        join hierarchy h
        on h.child_id = pc.grantee_name
        where not array_contains(pc.name::variant ,h.path)
        and granted_on = 'ROLE'
        and granted_to = 'ROLE'
        and privilege = 'USAGE'
        and deleted_on is null) ,
        final as (
        select distinct GRANTEE_NAME,
        'INSERT INTO ` + v_driving_table +` VALUES (\\''||GRANTEE_NAME||'\\', current_timestamp(0));' as INSERT_U,
        'ALTER USER '''||GRANTEE_NAME||''' SET NETWORK_POLICY='||` + v_network_policy + `' as SET_P
        from hierarchy h
         join snowflake.account_usage.grants_to_users gtu
            on h.top_role = gtu.role
        where CHILD_ID='` + v_driving_role + `'
        and DELETED_ON IS NULL
        and GRANTEE_NAME like '%@LOREAL.COM'
        and level<3)
        select
        coalesce(INSERT_U,'DELETE FROM ` + v_driving_table +` WHERE USER_NAME=\\''||USER_NAME||'\\'')
        ,coalesce(SET_P,'ALTER USER '''||USER_NAME||''' UNSET NETWORK_POLICY ') as statement
        from final f
        full join ` + v_driving_table +` d
        ON f.GRANTEE_NAME=d.USER_NAME
        where f.GRANTEE_NAME is null
        or d.USER_NAME is null;
        ;`;


//Execute sql to get list of distinct Users
let v_return_users =  snowflake.execute( {sqlText: v_return_users_sql} );
```

```
//Looping through the Users
 while (v_return_users.next()) {



  let v_user_whitelisting = v_return_users.getColumnValue(2);
  snowflake.execute( {sqlText: v_user_whitelisting} )

    let v_user_table = v_return_users.getColumnValue(1);
  snowflake.execute( {sqlText: v_user_table} )

 }

 return "Success"

$$;

call OPEN_IP_ADMIN_SP('_TECH_OPEN_INTERNET_ACCESS_ROLE','BREAKGLASS_NET_POL','USER_OPEN_INTERNET_ACCESS');
select * from USER_OPEN_INTERNET_ACCESS;
```

To schedule this procedure using the below statements

```
CREATE OR REPLACE TASK HRDP_OPEN_INTERNET_TASK
  WAREHOUSE = HRDP_DBT_BASE_WH
  SCHEDULE = '60 MINUTE'
  AS
    call OPEN_IP_ADMIN_SP('_TECH_OPEN_INTERNET_ACCESS_ROLE','BREAKGLASS_NET_POL','USER_OPEN_INTERNET_ACCESS');
```

Below statement is to check whether the task ran successfully or when will be the next scheduled run.

```
    SELECT *
  FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY())
  ORDER BY SCHEDULED_TIME;
```

Step 3:

For DBT, the network policy has created with below. We have to apply this policy to DBT Service account.

ALTER USER <> SET NETWORK_POLICY=<DBT_POLICY>;

Step 4:

For GCP, the network policy has created with below. We have to apply this policy to GCP Service account.

ALTER USER <> SET NETWORK_POLICY=<GCP_POLICY>;

Step 5:

For Open Network Policy (**HRDP_OPEN_NETWORK_POLICY**), created the stored procedure (**OPEN_INTERNET_ACCESS_SP**) to pull the users list from _TECH_OPEN_INTERNET_ACCESS_ROLE role.

You can verify whether you are belonging to open network policy or not using the below query.

SHOW PARAMETERS LIKE 'network_policy' IN USER "STEFANIA.PEDERIVA@LOREAL.COM";

Till here these documents guide you how the IP whitelisting is implemented in Loreal Project.

Below are the steps to run the command in the order, which is already executed

1. ALTER USER <> SET NETWORK_POLICY=<DBT_POLICY>;
2. ALTER USER <> SET NETWORK_POLICY=<GCP_POLICY>;
3. For Open network policy, automatically the stored proc is creating the alter statement and executed for the users.

1. For Microsoft Power BI Ip’s, the stored proc is creating network policy automatically.

AAD_Provisioning:

We have a job in AzureAD to assign the users to IT roles. This job is accessing the snowflake over the security integration. We have applied the HRDP_OPEN_NETWORK_POLICY to the integration directly to avoid security error.

Note : For all the statements, the owner is Accountadmin.