Part 3: Snowflake Monitoring and Security Validation Reference

This part is not a checklist but provides sample queries for setting up security monitoring and verifying security settings.

- Some settings can be verified with Snowflake Support via a Support Request. Other customer-managed configurations can be monitored using Snowflake's views and system functions.
- Monitoring may require using SQL commands to query the Account Usage and Information Schema views provided in the Snowflake database. These views provide detailed information about activity in your Snowflake account. Be aware that data in your Account Usage views may take up to 45 minutes to sync, so any monitoring activities that require detection in less than 45 minutes should use Information Schema views, not Account Usage views. See these examples in the Snowflake product documentation to learn how to query Account Usage views.
- SQL samples in this section are provided to help identify where to source information in the platform. Samples and descriptions are not a replacement for Professional Services advice or a full reading of the product documentation. Samples are just that, and should not be relied on to operationalize the monitoring controls recommended in Part 2 of this document.
- Most of the SQL samples assume the following context:

```
use role accountadmin;

use database snowflake;

use schema account_usage;
```

A. Network and client connection monitoring reference

Network policies provide options for managing network configurations to the Snowflake service.

❖ *Verify connections logging in from specific networks* *[A-1]*

Description

Verify that connections expected to log in from specific networks are doing so.

Related documentation

- `LOGIN_HISTORY` view

Sample statement - event list

```
select event_timestamp, user_name, client_ip, reported_client_type
reported_client_version,
first_authentication_factor,
second_authentication_factor
 from login_history where error_message = 'INCOMING_IP_BLOCKED'
 order by event_timestamp desc;
```

Sample statement - aggregation

```
--IPs where login failures are most often coming from
select distinct client_ip, count(client_ip)
from login_history
where is_success='NO'
group by client_ip;
```

Sample statement - aggregation

```
--Most blocked IP addresses (this requires Network Policies to be configured)
select distinct client_ip as blocked_source_ip
,count (client_ip), user_name
,reported_client_type as driver
,first_authentication_factor as authn_type
from login_history
where error_message = 'INCOMING_IP_BLOCKED'
group by user_name, client_ip, reported_client_type, first_authentication_factor
order by count (client_ip) desc
```

❖ *Monitor Network Policies* [A-2]

Description

Monitor changes to Network Policies and associated objects.

Related documentation

- QUERY_HISTORY (view)
- CREATE NETWORK POLICY, ALTER NETWORK POLICY, DROP NETWORK POLICY

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type in ('CREATE_NETWORK_POLICY', 'ALTER_NETWORK_POLICY', 'DROP_NETWORK_POLICY')
or (query_text ilike '% set network_policy%' or
    query_text ilike '% unset network_policy%')
order by end_time desc;
```

❖ *Show Network and User-level Policy settings* *[A-3]*

Description

Use these queries to verify that the Allowed and Blocked IP Addresses lists match expected configurations. Use the `SHOW NETWORK POLICIES` command to list all network policies defined in the system.

Related documentation

- `DESCRIBE NETWORK POLICY`
- `NETWORK_POLICY` (parameter)
- `SHOW INTEGRATIONS`, `DESCRIBE INTEGRATION`

Sample statement

```
show network policies;
desc network policy <name>;

show parameters like 'network_policy' in account;
show parameters like 'network_policy' in user <username>;
show integrations;
desc integration <integration_name>;
```

❖ *Monitor for long sessions* *[A-4]*

Description

Monitor for client applications that are keeping sessions open longer than desired by policy. Also look for client applications that have the `CLIENT_SESSION_KEEP_ALIVE` connection parameter set to true, which allows the session to NEVER expire if the connection is left open.

Related documentation

- `SHOW PARAMETERS`

- QUERY_HISTORY (view)
- ALTER SESSION

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type = 'ALTER_SESSION' and query_text ilike '%client_session%'
order by end_time desc;
```

B. Users and roles monitoring reference

To simplify monitoring, Snowflake recommends designating a role that will be used for all user management tasks. User/role creation or modification activity from a user/role other than the designated one should be flagged as suspicious activity.

❖ *Monitor that all user and role grants originate from the designated user management role [B-1]*

Description

Snowflake recommends using a designated role for all user management tasks. Monitor that all user and role grants originate from this role, and that this role is only granted to appropriate users.

Related documentation

- QUERY_HISTORY (view)
- GRANT ROLE

Sample statement

```
select user_name, role_name, query_text
from query_history where execution_status = 'SUCCESS'
and query_type = 'GRANT'
order by end_time desc;
```

❖ *Monitor built-in admin roles activity [B-2]*

Description

Watch the logs for all instances of a user using the default Snowflake admin roles to ensure their use is appropriate. The default admin roles are:

- ACCOUNTADMIN
- SECURITYADMIN

- `USERADMIN`
- `SYSADMIN`

Related documentation

- `QUERY_HISTORY` (view)

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and role_name in ('ACCOUNTADMIN','SECURITYADMIN','USERADMIN','SYSADMIN')
order by end_time desc;
```

❖ *Monitor for user creation* *[B-3]*

Description

Monitor `QUERY_HISTORY` for unusual `CREATE` or `REPLACE` user activity.

Related documentation

- `CREATE USER`

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type = 'CREATE_USER' and query_text ilike '%create%user%'
order by end_time desc;
```

❖ *Monitor for user modification* *[B-4]*

Description

Monitor `QUERY_HISTORY` for `ALTER user` activity, for example to flag non-SSO authentication method grants.

Related documentation

- `ALTER USER`

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type = 'ALTER_USER' and query_text ilike '%alter user%set rsa_public_key%'
order by end_time desc;
```

❖ *Monitor for re-enabling disabled users* [B-5]

Description

Re-enabling a user is a rare event that could be a threat. Monitor `QUERY_HISTORY` for `ALTER USER` activity.

Related documentation

- `ALTER USER`

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type = 'ALTER_USER' and query_text ilike '%alter user%set disabled = false%'
order by end_time desc;
```

❖ *Monitor for enabled plaintext user passwords* [B-6]

Description

Monitor `QUERY_HISTORY` for password changes.

Related documentation

- `ALTER USER`

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type = 'ALTER_USER' and query_text ilike '%alter user%set password%'
order by end_time desc;
```

❖ *Monitor SCIM user-provisioning API calls* *[B-7]*

Description

Applicable if SCIM user-provisioning via the REST API is configured. Monitor SCIM API calls to ensure API requests comply with policy.

Related documentation

- Auditing with SCIM

Sample statement

```
select *
    from table(snowflake.information_schema.rest_event_history(
        'scim',
        dateadd('minutes',-5,current_timestamp()), --change units and/or number
        current_timestamp(),
        200))
    order by event_timestamp desc;
```

❖ *Highly-privileged database object monitoring* *[B-8]*

Description

Monitor the following global privileges in `QUERY_HISTORY` because they involve elevated privileges in your Snowflake Account:

```
create user
create role
manage grants
create integration
create share
create account
monitor usage
OWNERSHIP
```

Sample statement

```
select user_name, role_name, query_text
from query_history where execution_status = 'SUCCESS'
and query_type = 'GRANT' and (
query_text ilike '%create user%' or
query_text ilike '%create role%' or
query_text ilike '%manage grants%' or
query_text ilike '%create integration%' or
query_text ilike '%create share%' or
query_text ilike '%create account%' or
query_text ilike '%monitor usage%' or
query_text ilike '%ownership%')
order by end_time desc;
```

❖ *Monitor `ACCOUNTADMIN` role grants* [B-9]

Description

The Snowflake role `ACCOUNTADMIN` should be closely monitored. Monitor [`QUERY_HISTORY`]() for `GRANT ROLE`.

Related documentation

- [Howto: How to review user's historical and current access grant role history]()

Sample statement

```
select user_name, role_name, query_text
from query_history where execution_status = 'SUCCESS'
and query_type = 'GRANT' and
query_text ilike '%grant%accountadmin%to%'
order by end_time desc;
```

❖ *Monitor grants to the public role* [B-10]

Description

The *public* role should have the fewest possible grants. Every user in a Snowflake account has *public*. Monitor [`QUERY_HISTORY`]() for alterations or grants to the public role.

Related documentation

- [FAQ: Why can all the roles access an object even though access has been granted to a single role?]()

Sample statement

```
select user_name, role_name, query_text, end_time
from query_history where execution_status = 'SUCCESS'
and query_type = 'GRANT' and
```

```
query_text ilike '%to%public%'
order by end_time desc;
```

C. Authentication monitoring reference

If you enable federated SSO for your Snowflake account, Snowflake recommends monitoring and pulling audit logs from both the identity provider and your Snowflake account.

❖ *Monitor how a user has authenticated* [C-1]

Description

The following sample statement shows the number of times each user authenticated and the authentication method they used.

Related documentation

- LOGIN_HISTORY
- How to identify if a user is using password or external based authentication

Sample statement

```
--Each User and their most frequently used authentication methods
select user_name,
first_authentication_factor,
second_authentication_factor, count(*)
 from login_history
 where is_success = 'YES'
 group by user_name, first_authentication_factor,
second_authentication_factor
order by user_name, count(*) desc;
```

❖ *Monitor if users who have used SSO before are using other authentication methods instead* [C-2]

Description

After users successfully authenticate using SSO, they should not be using other methods.

Related documentation

- LOGIN_HISTORY view

Sample statement

```
--login events for users who have sso but are not using it
select sso.*, event_timestamp, user_name, first_authentication_factor,
second_authentication_factor, client_ip, reported_client_type
reported_client_version
from login_history l
join (select user_name user_has_used_sso, min(event_timestamp) firstsso
        from login_history
       where first_authentication_factor in ('SAML2_ASSERTION','OAUTH_ACCESS_TOKEN')
        group by user_name) sso on sso.user_has_used_sso = l.user_name
where first_authentication_factor not in ('SAML2_ASSERTION','OAUTH_ACCESS_TOKEN')
and l.event_timestamp > firstsso
order by l.event_timestamp desc;
```

❖ *Monitor for Key Pair authentication* [C-3]

Description

Monitor the use of key pair authentication by querying login attempts.

Related documentation

- LOGIN_HISTORY view

Sample statement

```
select event_timestamp, user_name, client_ip, reported_client_type
reported_client_version,
first_authentication_factor,
second_authentication_factor
 from login_history
 where first_authentication_factor = 'RSA_KEYPAIR'
 order by event_timestamp desc;
```

❖ *Monitor if exclusive Key Pair authentication users are configured to use other authentication methods* [C-4]

Description

Users who have key pair authentication should be using it exclusively.

Related documentation

- ACCOUNT_USAGE » USERS view

Sample statement

```
--Key Pair Users who also have passwords
select * from users
where has_rsa_public_key = 'true'
and has_password = 'true';

--Key Pair Users who authenticated in a different way, and how many times
select u.name,
first_authentication_factor,
second_authentication_factor, count(*)
from login_history l
join users u on l.user_name = u.name and has_rsa_public_key = 'true'
where is_success = 'YES' and first_authentication_factor != 'RSA_KEYPAIR'
group by name, first_authentication_factor,
second_authentication_factor
order by count(*) desc;
```

❖ *Monitor for anomalous login activity* *[C-5]*

Description

Identifying Users who login frequently can help spot anomalies or unexpected behavior

Sample statement

```
--Most frequently authenticated Users (looking for anomalies)
select distinct user_name, count(user_name), first_authentication_factor
from login_history
where is_success = 'YES'
group by user_name, first_authentication_factor
order by count(user_name) desc;
```

❖ *Monitor for SCIM access token creation* *[C-6]*

Description

SCIM access tokens have a six-month lifespan so it is important to track how many were generated.

Related documentation

- SYSTEM$GENERATE_SCIM_ACCESS_TOKEN (Function)

Sample statement

```
select *

from query_history where execution_status = 'SUCCESS'

and query_text ilike '%system$generate_scim_access_token%';
```

❖ *Monitor failed login attempts* *[C-7]*

Description

The following approach returns results based on either the `FAILED_LOGINS` count or the login failure rate (`AVERAGE_SECONDS_BETWEEN_LOGIN_ATTEMPTS`). This approach helps distinguish a brute force attack from a human who is struggling to remember their password. There are inline comments on how to adjust the query to limit results.

Related documentation

- ACCOUNT_USAGE » LOGIN_HISTORY

Sample statement

```
select user_name,
       count(*) as failed_logins,
       avg(seconds_between_login_attempts) as
average_seconds_between_login_attempts
from (select user_name,
             timediff(seconds, event_timestamp, lead(event_timestamp)
                 over(partition by user_name order by event_timestamp)) as
seconds_between_login_attempts
      from account_usage.login_history
      where event_timestamp > date_trunc(month, current_date)
      and is_success = 'NO'
     )
group by 1
having count(*) > 3 //number of failures - adjust as needed
```

```
and avg(seconds_between_login_attempts) < 5 //average seconds between failures -
adjust as needed
order by avg(seconds_between_login_attempts) desc;
```

Sample statement (aggregation)

```
--Users who fail authentication most frequently
select distinct user_name, count(user_name), first_authentication_factor
from login_history
where is_success = 'NO'
group by user_name, first_authentication_factor
order by count(user_name) desc;
```

Monitoring Multi-Factor Authentication (MFA)

Either Snowflake or a third-party identity provider can enforce MFA.

❖ *Monitor Snowflake MFA* *[C-8]*

Description

Monitor the `SECOND_AUTHENTICATION_FACTOR` field in `LOGIN_HISTORY` to get per-user Snowflake MFA login information.

Related documentation

- Snowflake Multi-Factor Authentication (MFA) documentation

Sample statement

```
-- MFA/Authentication Stats
select
first_authentication_factor,
second_authentication_factor, count(*)
 from login_history where is_success = 'YES'
 group by first_authentication_factor,
second_authentication_factor
order by count(*) desc;

-- Most recent logins without MFA
select
event_timestamp, user_name, client_ip, reported_client_type
reported_client_version,
first_authentication_factor,
second_authentication_factor
 from login_history
 where first_authentication_factor = 'PASSWORD'
and second_authentication_factor is null
 order by event_timestamp desc;
```

❖ *Monitor non-Snowflake MFA* *[C-9]*

Description

The IdP provides monitoring. Snowflake has no way to determine if a third-party MFA solution is being used.

D. Encryption and key management monitoring reference

For Tri-Secret Secure, you can disable access to your KMS (the KMS where your customer-managed key is stored) at any time to verify this mechanism is configured properly. This is at your discretion and does not require any involvement from Snowflake. To validate, verify that any query of Snowflake data in the environment fails.

❖ *Verify that annual rekeying is enabled* *[D-1]*

Description

Annual rekeying is a best practice.

Related documentation

- `PERIODIC_DATA_REKEYING` (parameter)

Show Current State

```
show parameters like '%periodic_data_rekeying%' in account;
```

❖ *Monitor for any changes to the periodic_data_rekeying setting* *[D-2]*

Description

Changes to this setting are rare and deserving of scrutiny.

Related documentation

- `PERIODIC_DATA_REKEYING` (parameter)

Sample statement

Monitor `QUERY_HISTORY` for queries matching the pattern:

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS'
and query_type = 'ALTER_ACCOUNT' and query_text ilike '%periodic_data_rekeying%'
order by end_time desc;
```

E. Integration monitoring reference

Validate your integrations and document how they are configured. Once validated, monitor the `QUERY_HISTORY` view for queries like:

```
'%integration%';
```

Because integrations can enable a new means of access to Snowflake data, closely monitor for new integrations or the modification of existing integrations.

Note	See also "To monitor SCIM user-provisioning API calls" in the "Users and roles monitoring reference."

❖ *Show integrations in Snowflake* [E-1]

Description

Integrations should be periodically reviewed.

Related documentation

- `SHOW INTEGRATIONS` command

Sample statement

```
show integrations;

desc integration <name>;
```

❖ *Monitor for security integrations that have been created or altered* [E-2]

Description

Security integrations shouldn't be added or changed on a frequent basis. Monitor `QUERY_HISTORY` for alterations

Related documentation

- `CREATE SECURITY INTEGRATION`

Sample statement

```
select end_time, query_type, query_text, user_name, role_name
from query_history where execution_status = 'SUCCESS' and
query_type in ('CREATE','ALTER') and
query_text ilike '%security integration%'
order by end_time desc;
```

F. Backup and recovery monitoring reference

Snowflake Time Travel enables accessing historical data (*e.g.* data that has been changed or deleted) at any point within a defined period.

Monitoring data retention (Time Travel)

Any changes to retention time should be monitored on any objects containing sensitive objects. Retention time is inherited from parent objects, so altering retention time at the Database level, for example, would also alter retention time of the objects therein.

❖ *Monitor data retention* *[F-1]*

Description

There is retention at different levels (database, schema, account). Monitor `QUERY_HISTORY` for queries matching the following pattern for all sensitive objects for any deviation from the standard determined by your business requirements as they could indicate that persisted data may be used for unintended purposes.

For a full audit of retention policy, the `TABLES` view in Account Usage provides a field called `RETENTION_TIME`.

Related documentation

- `DATA_RETENTION_TIME_IN_DAYS` (parameter)

Sample statement(s)

```
show parameters like '%data_retention_time_in_days%' in account;
select schema_name, catalog_name, schema_owner, retention_time from schemata;
select database_name, database_owner, retention_time from databases;
select table_name, table_schema, table_catalog, table_owner, table_type,
retention_time from tables;
```

G. Snowflake parameter monitoring reference

❖ *Monitor account-level parameters* *[G-1]*

Description

Snowflake recommends monitoring the following account-level parameters for security purposes.

- `ALLOW_ID_TOKEN`
- `CLIENT_ENCRYPTION_KEY_SIZE`
- `INITIAL_REPLICATION_SIZE_LIMIT_IN_TB`
- `NETWORK_POLICY`
- `PERIODIC_DATA_REKEYING`
- `PREVENT_UNLOAD_TO_INLINE_URL`
- `REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_CREATION`
- `REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_OPERATION`
- `SSO_LOGIN_PAGE`

Related documentation

- Parameters reference documentation

Sample statement

```
show parameters like '%allow_id_token%' in account;
show parameters like '%client_encryption_key_size%' in account;
show parameters like '%initial_replication_size_limit_in_tb%' in account;
show parameters like '%network_policy%' in account;
show parameters like '%periodic_data_rekeying%' in account;
show parameters like '%prevent_unload_to_inline_url%' in account;
show parameters like '%require_storage_integration_for_stage_creation%' in account;
show parameters like '%require_storage_integration_for_stage_operation%' in account;
show parameters like '%sso_login_page%' in account;
```

❖ *Monitor SAML and SSO account-level parameters* *[G-2]*

Description

SAML and SSO parameter changes should be a rare event.

Sample statement

```
show parameters like 'saml%' in account;
show parameters like 'sso%' in account;
```

H. Snowflake Access History monitoring reference

❖ *Monitor Access History* *[H-1]*

Description

Monitor User access history ordered by user and query start time, starting from the most recent access.

Sample statement

```
select user_name
, query_id
, query_start_time
, direct_objects_accessed
, base_objects_accessed
from access_history
order by 1, 3 desc;
```

Description

Add the `object_id` value to determine who accessed a sensitive table in the last 30 days.

Sample statement

```
select distinct user_name
from access_history
     , lateral flatten(base_objects_accessed) f1
where f1.value:"objectId"::int=<fill_in_object_id>
and f1.value:"objectDomain"::string='Table'
and query_start_time >= dateadd('day', -30, current_timestamp());
```