Security Features Checklist

Use this document to review your security choices to ensure important Snowflake settings have not been overlooked. Security monitoring recommendations are also provided.

Note:	This document is not a substitute for a thorough, services-led deployment, nor does it supersede any other obligations you may have to Snowflake, your organization, your outside regulators, or other bodies to which you owe compliance or conformance. The checklists and code samples contained in this document are informational only.

This document has three parts:

- Part 1: Security Features Enablement Checklist helps you make informed choices about the security features included with your Snowflake deployment
- Part 2: Security Monitoring Checklist provides monitoring recommendations
- Part 3: Snowflake Monitoring and Security Validation Reference is not a checklist but documents sample queries for monitoring security and validating security settings

Part 1: Security Features Enablement Checklist

The following sections are intended to help you make informed choices about the security features included with your Snowflake deployment. To complete the checklist, select an option in each section.

A. All Editions Checklist

A0. Choose the Snowflake "Account Name" and "Organization Name." Be advised that DNS name(s) incorporate both your Account Name and Organization Name. Because DNS names for Snowflake accounts are publicly visible, choose names that satisfy your organization's policies about naming publicly viewable assets. Snowflake creates at least two DNS names per account when the account is created. Each DNS name will have a different account identifier that either Snowflake or your Snowflake admin assigns, and an account locator identifier, which is a random name that Snowflake assigns when the account is created. If you are using Snowflake Business Critical Edition (or higher) and you need alternate DNS names for other connectivity, consider using Connections. Connections function as alternate "Account Name" values that you can use for scenarios involving alternate DNS names for Snowflake account access (*e.g.* Client Redirect / Failover).	
☐	*Obfuscate your organization name when choosing Organization and Account Names. This is required if your policy dictates that. If you are using Snowflake Business Critical Edition (or higher) and you require alternate DNS names for other connectivity, use Connections.*
✔	*Do not obfuscate your organization when choosing Organization and Account Names. If you are using Snowflake Business Critical Edition (or higher) and you require alternate DNS names for other connectivity,* use *Connections.*

A1. Choose a Network Policy (Allowed/Blocked IP Addresses lists) strategy. Decide how to use the built-in Network Policy feature to either lock out all-but-certain IP ranges (Allowed IP Addresses list), or lock out specific ranges (Blocked IP Addresses list). See the Network Policy section of the documentation for details. For an example of how specific use case Network Policies are used, refer to the Okta-specific SCIM Network Policy documentation. For other use cases, consult your specific service's documentation.	
☐	*No Network Policy.* No extra setup required.
☐	*Apply a Network Policy for specific users or services, but do not apply an account-wide Network Policy.* Requires setting up the specific use-case Network Policy.
✔	*Apply both a Network Policy for specific users or services, and an account-wide Network Policy.* Requires setting up the specific use-case Network Policies *and* the account-wide Network Policy.

A2. Choose how to provision users. Choose a method through which users will be created in Snowflake. Snowflake recommends using some form of automated user creation whenever this is possible.	
✔	*Use SCIM.* Requires setup in your authoritative user management system. See the SCIM documentation for specific setup instructions. This can be combined with SQL driven.
☐	*Use SQL-driven automation.* Requires setup in your authoritative user management system. To understand the full scope of manageable features through your system, please refer to the access control documentation. This can be combined with SCIM.
☐	*Create users manually.* Not recommended.

A3. Choose how to manage SCIM tokens. To use SCIM, you must create and save the authorization token. Use this token for each SCIM REST API request and place it in the request header. SCIM providers like Azure AD, Okta, and other commercial products will have a place to save this token so that they can access Snowflake to perform the SCIM operations. The access token expires after six months. A new access token can be generated using SQL. You should have a plan in place to automate or manage this lifecycle.	
☐	*Use automation to manage the SCIM access token lifecycle.* This is recommended, and you can then choose your own cycle (hopefully shorter than 6 months).
✔	*Use a manual process to manage the SCIM access token lifecycle.* Not recommended.

A4. Review session keep-alive timings. There are two settings that affect the length of time a session will be kept alive for some Snowflake clients based on the following drivers/connectors: Python, SnowSQL, JDBC, and Node.js. The settings are `CLIENT_SESSION_KEEP_ALIVE` and `CLIENT_SESSION_KEEP_ALIVE_HEARTBEAT_FREQUENCY`.	
☐	*Review the values of these settings.*
☐	*Accept the defaults.*

A5. Use SAML-based single sign-on (SSO). The best practice for human users is to use SSO to authenticate with an authoritative IdP (Identity Provider) or directory (*e.g.* Active Directory).	
✔	*Use SSO.* Requires SSO setup in Snowflake and your IdP. See the SSO documentation.
	Do not use SSO. Not recommended. Skip to A6.

A5.1 Check your SAML NameID Format. A common customization that many organizations make to their SAML configuration is using a custom NameID Format (*e.g.* using a corporate ID instead of the default email address as the login name). If this describes your organization, ensure that your Snowflake configuration has been adjusted accordingly.	
	Align Snowflake settings with your organization's SAML NameID Format. Requires SSO setup in Snowflake. See the SSO documentation.
	Your organization's SAML NameID Format is Email. No action needed.

A5.2 Use SAML assertion encryption. If your policy requires SAML assertions to be encrypted, ensure that your Snowflake configuration accounts for that.	
	Change Snowflake settings to encrypt SAML assertions. Requires SSO setup in Snowflake. See the SSO documentation.
	Your organization's policy doesn't require SAML assertion encryption. No action needed.

A5.3 Use Signed SAML Requests. If your policy requires SAML requests to be signed, ensure that your Snowflake configuration accounts for that.	
	Change Snowflake settings to sign SAML requests. Requires SSO setup in Snowflake. See the SSO documentation.
	Your organization's policy doesn't require SAML request signing. No action needed.

A5.4 Use a Custom Logout Endpoint. If your policy requires Custom Logout Endpoint for SAML, ensure that your Snowflake configuration accounts for that.	
☐	*Change Snowflake settings to use a Custom Logout Endpoint for SAML.* Requires SSO setup in Snowflake. See the SSO documentation.
☐	*Your organization's policy doesn't require a Custom Logout Endpoint for SAML.* No action needed.

A5.5 Activate ALLOW_ID_TOKEN in combination with SAML and External Browser mode. Activating the ALLOW_ID_TOKEN parameter prevents desktop applications that begin multiple sessions in the background from popping up repeated authentication requests to the users. This setting applies when using External Browser mode with Snowflake drivers on desktops to allow desktop applications that do not support SAML natively to authenticate using SAML.	
☐	*Change Snowflake settings to permit the ALLOW_ID_TOKEN.* Requires SSO setup in Snowflake, and a driver on a desktop using External Browser mode.
☐	*Do not use ALLOW_ID_TOKEN.* No action needed.

A6. Use key pair authentication. This authentication method is typically used with service accounts and other non-human entities that establish automated connections to Snowflake. For an example of how this may be leveraged see the JDBC driver Key Pair configuration documentation.	
☐	*Use Key Pair authentication.* Requires setup of your connection for each use case where it will be leveraged. After identifying where you will use it, refer to the specific driver documentation to find the specific configuration for that driver.
☐	*Do not use Key Pair authentication.* No extra setup required.

A7. Evaluate if you need built-in passwords for users. Though Snowflake does not encourage using built-in passwords for users, it is not uncommon for tools that need to access Snowflake-hosted information to only work with username/password authentication. Identify this requirement early and figure out which users will need to have a password in Snowflake, then use the best approach possible to manage that password. If you must use built-in passwords, consider using built-in MFA (see **A8**) and user-level Network Policies (see **A9**) for added protection.	
✔	*Do not use built-in passwords.* This is the best practice.
	Assign built-in passwords selectively. If you can isolate a subset of your total users that require access to Snowflake with a username and password, then it's best to assign only these users a built-in password. This could be driven by a provisioning or access governance system or process (*e.g.* put in a policy stating that any user assigned tool "X" that requires a username and password will trigger a built-in password being assigned to the Snowflake user).
	Assign built-in passwords for all users. This is not recommended but may be required by other platforms that are unable to support better options when authenticating to Snowflake.

A8. Enable built-in MFA if you plan to use built-in passwords for users. Built-in Multi-Factor Authentication (MFA) only supports built-in passwords for users. (See **A7**.) If you use SSO, leverage the MFA from your Identity Provider. If you must use built-in passwords, consider using built-in MFA wherever possible. Compatibility with Snowflake's built-in MFA will depend on the tools used to access Snowflake, but most should be compatible based on Snowflake's design. For details on the built-in MFA, see the MFA documentation. Note that users must do a one-time registration to activate built-in MFA.	
	Use built-in MFA for all users with built-in passwords. Configure built-in MFA for all built-in password users.
	Use built-in MFA for select users with built-in passwords. Configure built-in MFA for tools and users that are MFA compatible, and isolate tools that are not compatible.
	Do not use built-in MFA. Not recommended.

A9. Enable user-level Network Policies if using built-in user passwords for users or service accounts. Snowflake offers user-level Network Policies as an added layer of protection if using built-in passwords (see **A7**). User-level Network Policies are especially effective at protecting service accounts. For more information, see Managing User-level Network Policies.	
☐	*No Network Policy.* No extra setup required.
☐	*Apply a Network Policy for a specific user or service.* Requires setting up the specific use-case Network Policy.

A10. Review built-in admin roles. Grant the built-in Snowflake administrative roles (`ACCOUNTADMIN`, `SECURITYADMIN`, `USERADMIN`, `SYSADMIN`) carefully to ensure a "least privilege" approach to administering Snowflake. These roles should never be any user's default role. Periodically review who should have these roles and why. Document your role settings and policies, and ensure that your rollout plan puts the deployment in line with these policies.	
☐	*Review admin role-grants before going live.*
☐	*Leave admin role-grants as-is before going live.* Not recommended.

A11. Review external stage configurations. External stages are Snowflake objects connected to cloud storage outside of Snowflake, which are used to move data into and out of the platform. (See the CREATE STAGE documentation for details.) You can control how users create these objects, and where in the cloud users may point them. Review how these will be created initially and on an ongoing basis. Make sure you understand how the following Snowflake parameters change external stage behavior: `REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_CREATION`, `REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_OPERATION`, `PREVENT_UNLOAD_TO_INTERNAL_STAGES`, and `PREVENT_UNLOAD_TO_INLINE_URL`.	
☐	*Review your organization's external stage creation process before going live.*
☐	*Leave your organization's external stage creation process as is before going live.* Not recommended.

A12. Decide if users can set/reset Snowflake built-in passwords. Any authenticated user would normally have the ability to set or reset the built-in password for their Snowflake user account. If you do not want users to have this ability, set the `DISABLE_SELF_PASSWORD_CHANGE` parameter for the Snowflake account. See the User Management documentation for more details. (Note: users who do not have any built in password will not be able to use this feature to assign themselves one.)	
☐	*Review user password reset settings before going live.*
☐	*Leave user password reset settings as is before going live.* Not recommended.

A13. Consider using Managed Access Schemas to further lock down object security by removing owner grant-rights. In a managed access schema, object owners lose the ability to make grant decisions. Only the schema owner (*e.g.* the role with the `OWNERSHIP` privilege on the schema), or a role with the `MANAGE GRANTS` privilege can grant privileges on objects in the schema, including future grants, centralizing privilege management. Otherwise, with non-managed schemas in a database, object owners (*e.g.* roles with the `OWNERSHIP` privilege on one or more objects) can grant object access to other roles, with the option to further grant those roles the ability to manage object grants. For more information on managed access schemas, see the Creating Managed Access Schemas documentation.	
☐	*Review owner grant-rights role models before going live.*
☐	*Do not review owner grant-rights role models before going live.* Not recommended.

A14. Register to receive notifications about service outages. Snowflake administrators and business users can go to status.snowflake.com and click **Subscribe to Updates** to receive near real-time notifications about service outages. Notifications are specific to Snowflake and supported Cloud Service Provider regions.	
☐	*Register at status.snowflake.com to receive notifications about service outages.*
☐	*Do not register users to receive service updates.* Not recommended.

A15. Use the "ALL" role as a Secondary Role to avoid managing "personal roles." Snowflake supports the ability for users to have all of their role grants active in their session instead of just a single role, allowing for visibility of all objects and data without switching roles. This is especially useful with Row Access Policies, so users can see all the rows they are authorized to see in a single query.	
☐	*Alter Users to have a SECONDARY_ROLE = 'ALL'*
☐	*Continue to have Users utilize a single role per session.*

B. Enterprise Edition and Higher Checklist

The following applies to *Enterprise* and higher editions. The All Editions checklist also applies.

B1. Adjust Time Travel settings & ensure policy alignment for retention. Time Travel can be set from 0 (off) to 90 (max) days for each object (*e.g.* Table). The default setting is 1 day. Fail-safe also provides a 7-day recovery window on top of that. Your legal or auditing groups may have important views with regards to the policy impact of how long this data may be recovered (in the context of GDPR, for example). Snowflake recommends reviewing the plans for these features with those groups to ensure policy alignment. • See the Time Travel documentation for details about those settings. • See the Fail-safe documentation for details about that feature.	
☐	*Review Time Travel & Fail-safe Settings with Policy Groups.* Recommended.
☐	*Do Not Review Time Travel & Fail-safe Settings with Policy Groups.* Not recommended.

B2. Activate yearly re-keying. Enterprise edition or higher customers can choose to have Snowflake replace all Table Master Keys (TMK) with new keys on an annual basis. This security best practice ensures that the scope of any one key does not exceed a period of one year. For more details, see the Data Encryption documentation. To enable this feature, run this SQL as the `ACCOUNTADMIN` role: `` `alter account set PERIODIC_DATA_REKEYING=TRUE;` ``	
✔	*Activate Yearly Re-keying.* Recommended.
☐	*Do Not Activate Yearly Re-keying.* Not recommended.

B3. Use Session policies. *Note: This is a Preview feature.* A session begins when a user connects to Snowflake and successfully authenticates. The default idle session timeout period is 4 hours. To configure a custom session idle timeout period, implement a session policy and define an alternate idle session timeout period in minutes. For more information see Session Policies.	
☐	*Use default idle timeout of 4 hours.*
☐	*Alter idle session timeout value.*

B4. Use column-level security as part of your data governance strategy. Column-level security in Snowflake applies a masking policy to a column within a table or view.

Column-level security comprises two features:

- Dynamic data masking, which includes column-level encryption, data masking, hashing, and obfuscation.
- External tokenization, which tokenizes sensitive data before it is loaded into Snowflake, and dynamically detokenizes the data at query runtime.

Column-level policies can be created:

- With context functions
- With conditional expression functions
- With mapping tables
- With user defined functions

✔	*Use column-level security.*
	Do not use column-level security. Skip to B5.

B4.1 Apply additional column-level encryption/decryption. Applies if using column-level security. Match conditions using tags, context functions, entitlement tables, or user-defined functions (UDFs), then *encrypt*/*decrypt* based on data sensitivity.

	Yes, and we will manually supply encryption keys at query time. Requires column-level security.
	Yes, and we will programmatically supply keys at runtime via a Snowflake external function. Requires column-level security. Note: With this option, Snowflake does not manage your encryption keys. Instead, keys are hosted in your environment—in your HSM, for example.To retrieve the keys at run time, use an external function.
	No column-level encryption required.

B4.2 Apply data masking. Outputs `_masked_` or `******` or something else depending on the masking scheme. Match conditions using tags, context functions, entitlement tables, or user-defined functions (UDFs), then apply masking based on data sensitivity.	
☐	*Use data masking.* Requires column-level security.
☐	*Do not use data masking.*

B4.3 Apply hashing with salt. Match conditions using tags, context functions, entitlement tables, or user-defined functions (UDFs), then apply hashing based on data sensitivity.	
☐	*Apply hashing and we will manually supply salts at query time.* Requires column-level security.
☐	*Apply hashing and we will programmatically supply salts at runtime via a Snowflake external function.* Requires column-level security. Note: With this option, salts *are* generated in your environment—for example, using your HSM. To retrieve the salts at run time, use an external function.
☐	*No column-level hashing required.*

B4.4 Apply Obfuscation. For example, if the data within a field has an expected format, such as a Social Security number, return a substring of the field.	
☐	*Apply obfuscation using built-in functions like string functions, concat, and so on.* Requires column-level security.
☐	*Apply obfuscation using user-defined functions.* Requires column-level security.
☐	*No obfuscation required.*

B4.5 Use External Tokenization. Tokenize sensitive data before loading it into Snowflake, and dynamically detokenize data at query runtime using masking policies with External Functions.	
☐	*Use an external tokenization provider.* Requires column-level security.
☐	*No tokenization required.*

B5. Use row access policies as part of your data governance strategy. Snowflake supports row-level security through the use of row access policies to determine which rows to return in the query result. The row access policy can be relatively simple, for example, to allow one particular role to view rows; or it can be more complicated, for example, by including a mapping table in the policy definition to determine access to rows in the query result. Select one or more of the following choices.	
☐	*Create row access policies with context functions to determine which rows to return in the query result.*
☐	*Create row access policies with conditional expression functions to determine which rows to return in the query result.*
☐	*Create row access policies with mapping tables to determine which rows to return in the query result.*
☐	*Create row access policies with user defined functions to determine which rows to return in the query result.*
☐	*Do not use row access policies.*

B6. Use tagging as part of your data governance strategy. Tags help data stewards track sensitive data for compliance, discovery, protection, and resource usage use cases through either a centralized or decentralized data governance management approach. For more information, see Using tags with Snowflake objects and features.	
✔	*Use tags to track sensitive data stored in Snowflake.*
☐	*I already have external tagging systems or I do not plan to use tags.*

B6.1 Use tag-based masking policies *Note: This is a Preview feature.* Tag-based masking policies allow policy administrators to simplify data protection efforts by applying one or more masking policies to a tag, and then applying the tag to many objects. For more information, see Tag-Based Masking Policies.	
☐	*Use tag-based masking policies with the tags assigned in B6.*
☐	*I already have external tagging systems or I do not plan to use tags.*

B7. Use classification as part of your data governance strategy. *Note: This is a Preview feature.* Classification helps you identify and manage personal and/or sensitive data stored in Snowflake to ensure the data is not exposed. Classification uses tags to categorize and label data. Use the tags to facilitate analysis and compliance with privacy regulations. For more information, see Data Classification Functions and Stored Procedures to discover and apply tags using data classification. Check the classification categories.	
☐	*Use classification to identify and manage personal and/or sensitive data stored in Snowflake.*
☐	*Do not use classification*.

C. Business Critical Edition and Higher Checklist

The following applies to *Business Critical* and higher editions. The All Editions and Enterprise checklists also apply.

C1. Decide if user connections will use a public or private endpoint for network connectivity. Decide the network connectivity for the majority of use cases. Will user connections use a public endpoint (the default), or some form of private endpoint (requires setup and integration of cloud service provider private networking)?	
☐	*Use a public endpoint.* No extra setup required. [Most common choice]
☐	*Use a mix of public and private endpoints.* Requires setup and integration of cloud service provider private networking. May also require Network Policies to do IP allow-listing based on specific requirements. [About 10% choose this]
☐	*Use a private endpoint exclusively.* Requires setup and integration of cloud service provider private networking. Requires Network Policies to do IP allow-listing to lock out all but private IP ranges coming through the private endpoint. [About 1% choose this]

C2. Activate Tri-Secret Secure. Tri-Secret Secure is the name of the Bring Your Own Key (BYOK) feature in Snowflake. To configure Tri-Secret Secure, create a key in your cloud provider's key management system, then contact Snowflake support and provide the metadata about that key to start the process of activating the feature. For more details about BYOK see the Tri-Secret Secure documentation.	
☐	*Activate Tri-Secret Secure.*
☐	*Do Not Activate Tri-Secret Secure.*

C3. Enable multi-region database replication for high availability support. For details, see the Database Replication and Failover/Failback documentation.	
☐	*Enable database replication between Snowflake accounts (within the same organization) to keep database objects and stored data synchronized.*
☐	*Do not replicate database objects across regions.*

C4. Enable database failover/ failback to support disaster recovery and business continuity scenarios. For details, see the Database Replication and Failover/Failback documentation.	
☐	*Activate database failover and failback.*
☐	*Do not activate failover and failback to replicated databases.*

C5. Enable Client Redirect. When enabled, Client Redirect redirects your client connections to Snowflake accounts in different regions, either for business continuity and disaster recovery purposes, or if migrating your account to another region or cloud platform. To use Client Redirect, both the Primary and Secondary accounts need to be Business Critical Edition or higher. Reader Accounts are currently not supported.	
☐	*Enable Client Redirect.*
☐	*Do Not Enable Client Redirect.*

C6. Decide if private connectivity to the internal stage is needed (AWS and Azure). By default, the cloud service provider's public endpoint is used for internal stage connectivity. GCP does not yet support private connectivity to the internal stage.	
☐	*Use AWS S3 PrivateLink or Azure Blob Storage Private Endpoint for internal stage connectivity.*
☐	*Private connectivity to the internal stage is not required or not available. We will use the cloud service provider public endpoint.*

Part 2: Security Monitoring Checklist

The following items are the *minimum* amount of monitoring Snowflake recommends for effective security controls. Use this list like a checklist or a to-do list. The more items you check off the better.

A. Network, Network Policy, and client connection monitoring items. Monitor the following to ensure compliance with policy. See "Network and client connection monitoring reference" in part 3 for help with monitoring queries.	
☐	**1. Built-in network connectivity.** Verify that all connections expected to log in from specific networks are doing so. To check, query the `LOGIN_HISTORY` objects.
☐	**2. Network Policy configuration.** Check if all Allowed and Blocked IP Addresses lists match expected configurations. To check, use `SHOW NETWORK POLICIES` and `DESCRIBE NETWORK POLICY` SQL statements.
☐	**3. Long sessions.** Monitor for client applications that are keeping sessions open longer than desired by policy. To do so, 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.

B. Users and roles monitoring items. Monitor the following to ensure compliance with policy. See "Users and roles monitoring reference" in part 3 for help with monitoring queries.	
☐	**1. Designated user-management role activity.** To simplify monitoring, 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.
☐	**2. Built-in admin roles activity.** Watch the logs for all instances of a user using the built-in admin roles (`ACCOUNTADMIN`, `SECURITYADMIN`, `SYSADMIN`, `USERADMIN`) to ensure their use is appropriate.
☐	**3. User creation or modification.** Monitor `QUERY_HISTORY` for unusual `CREATE` or `REPLACE` user activity.
☐	**4. User activity.** Monitor audit logs, which contain audit trails with all the activity of your users. See the Account Usage documentation for details about what information is available and how to obtain it.
☐	**5. SCIM user-provisioning API activity.** If SCIM user-provisioning is configured, monitor SCIM API calls to ensure API requests comply with policy.

☐	**6. Roles with access to sensitive fields.** Monitor any role with read access to schemas/tables containing non-obfuscated sensitive fields, such as those containing PII information or confidential metrics. Perform a regular audit of your monitoring rules to ensure they encompass all appropriate custom roles in your Snowflake environment.
☐	**7. Grants to the public role.** The *public* role should have the fewest possible grants. Every user in a Snowflake account is granted the public role.

C. Authentication monitoring items. Monitor the following to ensure compliance with policy. See "Authentication monitoring reference" in part 3 for help with monitoring queries.	
☐	**1. SSO use.** If using SSO, query the `LOGIN_HISTORY` objects to check if all expected SSO users actually use it. This is especially important if users have multiple authentication methods configured but policy requires the use of SSO whenever possible.
☐	**2. Key Pair use.** Since service accounts and other automated systems with sensitive privileges mainly use Key Pair authentication, ensure that these users have not used other authentication methods that are not in compliance with policy. To do so, query the `LOGIN_HISTORY` objects to monitor users who use Key Pair authentication exclusively.
☐	**3. Key Pair configuration.** Monitor exclusive Key Pair authentication users to ensure they are not *configured to use* other authentication methods.
☐	**4. SCIM access tokens.** If SCIM user-provisioning via the REST API is configured, monitor for the creation of SCIM access tokens.
☐	**5. Failed logins.** Monitor for failed login attempts.
☐	**6. Built-in passwords configuration.** If using built-in passwords, query the users to verify that all users that have built-in passwords configured are expected to have it configured.
☐	**7. Built-in MFA configuration.** If using built-in MFA, query users to check if all users expected to have MFA configured actually do have it configured in order to ensure compliance with policy.

D. Encryption and key management monitoring items. Monitor the following to ensure compliance with policy. See "Encryption and key management monitoring reference" in part 3 for help with monitoring queries.	
☐	**1. Annual rekeying.** Verify that annual rekeying is enabled.
☐	**2. Scheduled data rekeying.** Monitor the Snowflake account for any changes to periodic data rekeying.

E. Integration monitoring items. Monitor the following to ensure compliance with policy. See "Integration monitoring reference" in part 3 for help with monitoring queries.	
☐	**1. Creation or modification of integrations.** Because integrations can enable a new means of access to Snowflake data, closely monitor for new integrations or the modification of existing integrations.

F. Backup and recovery monitoring items. Monitor the following to ensure compliance with policy. See "Backup and recovery monitoring reference" in part 3 for help with monitoring queries.	
☐	**1. Time Travel settings for critical objects.** After reviewing Time Travel settings with your policy makers, there will likely be guidelines for specific objects about what their Time Travel settings should be. Monitor all objects where these settings have policy impacts.

G. Snowflake parameter monitoring items. Monitor the following to ensure compliance with policy. See "Snowflake parameter monitoring reference" in part 3 for help.	
☐	**1. Account-level parameters.** Parameters allow customers to configure and further secure their environments. Account-level parameters should be monitored for security purposes.
☐	**2. SAML and SSO account-level parameters.** If you enable federated SSO for your Snowflake account, monitor SAML and SSO account-level parameters.

H. Evaluate internal log retention policies and compliance requirements. Snowflake maintains audit log information in a tamper-proof area of the database for 365 days.	
☐	**1. No additional log retention required.** Query using Snowflake drivers, connectors, or APIs, such as Snowsight, to return data as desired for reporting.
☐	**2. Export log tables into Snowflake.** Retain log table information in Snowflake indefinitely to enable longer term audit trails.
☐	**3. Transform and export logs.** Transform and export logs outside of Snowflake to CSP Storage Services for retention and reporting via a SIEM/SOAR or XDR Tool.

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());
```