Dremio Software

Security Best Practices

Introduction

This document describes the best practices for security in a Dremio environment. The document is intended for use by Dremio administrators and developers who are responsible for setting up and maintaining a Dremio cluster.

As there are different aspects of security implementation within a Data Lakehouse Platform, this document will be divided into three sections:

- Infrastructure Security, covering encryption practices and authentication options
- Data Security, explaining how RBAC works in Dremio, how privileges are inherited between objects, and how to develop granular controls over data access using User-Defined Functions
- Audit Logging, describing the type of logging that can be performed to ensure access to the data isn't abused and to enable easy and consistent auditing of the platform

Infrastructure Security

When administering a self-managed Dremio cluster, it is imperative to ensure that the environment follows good security practices to secure communications between servers and provide modern authentication methods.

Encryption

Wire encryption provides confidentiality and privacy to two parties communicating over a public network. The two parties may also need to prove their identity to each other: authentication is the process of proving identity.

Dremio supports the following TLS wire encryption methods:

- Full Wire Encryption - Enables all TLS communication.
- Web Server Encryption - Enables HTTPS on Dremio's web server.
- Encryption for Arrow Flight (Including the ODBC Driver for Arrow Flight SQL) - Enables TLS communication between Arrow Flight client applications and a Dremio cluster
- Encryption for JDBC Clients and Power BI Clients - Enables TLS communication between JDBC client applications and a Dremio cluster, or between Power BI client applications and a Dremio cluster.
- Intracluster Encryption - Enables TLS communication between nodes in a Dremio cluster.

Enabling Full Wire Encryption is recommended in most applications, especially when the cluster is not only accessible within a private enterprise network. At a minimum, a production environment will need to have Web Server Encryption and encrypt all communications between Dremio and the clients being used by end users.

How to enable Encryption

Encryption is typically enabled through specific parameters in dremio.conf and requires the keystore and truststore of the certificates to be available on the coordinator node. The following file permissions must be set to allow the new settings to be taken up correctly:

- keystore permission: 0440
- truststore permission: 0444
- dremio.conf file permission: 0444

Here are the configurations that need to be applied to dremio.conf to enable the various levels of encryption. Please note that if Full Wire Encryption is enabled, there is no need to enable the more granular levels of encryption.

If you have more than one coordinator node as part of your Dremio deployment, ensure you are changing this configuration **on all coordinators.**

Full wire encryption configuration

```
javax.net.ssl.keyStoreType: "type" # optional; default: JKS
javax.net.ssl.keyStore: "path/to/keystore/jks/file"
javax.net.ssl.keyStorePassword: "keystorePassword"
javax.net.ssl.keyPassword: "key password"
javax.net.ssl.trustStoreType: "type" # optional; default: JKS
javax.net.ssl.trustStore: "path/to/truststore/jks/file"
javax.net.ssl.trustStorePassword: "trustStorePassword"

services.coordinator.client-endpoint.ssl.enabled: true
services.coordinator.client-endpoint.ssl.auto-certificate.enabled: false
services.coordinator.web.ssl.enabled: true
services.coordinator.web.ssl.auto-certificate.enabled: false
services.fabric.ssl.enabled: true
services.fabric.ssl.auto-certificate.enabled: false
```

Web Server Encryption Configuration

```
services.coordinator.web.ssl.enabled: true
services.coordinator.web.ssl.auto-certificate.enabled: false
services.coordinator.web.ssl.keyStore: "path/to/keystore/jks/file",
services.coordinator.web.ssl.keyStorePassword: "keystorePassword",
services.coordinator.web.ssl.trustStore: "path/to/trustStore", (Optional)
services.coordinator.web.ssl.trustStorePassword: "trustStorePassword" (Optional)
```

Arrow Flight Client Encryption Configuration

```
services.flight.ssl.enabled: true
services.flight.ssl.auto-certificate.enabled: false
services.flight.ssl.keyStoreType: "jks"
services.flight.ssl.keyStore: "/path/to/serverKeyStore.jks"
services.flight.ssl.keyStorePassword: "<password for your keystore>"
services.flight.ssl.keyPassword: "<password for your key>"
services.flight.ssl.trustStoreType: "jks"
services.flight.ssl.trustStore: "/path/to/serverTrustStore.jks"
services.flight.ssl.trustStorePassword: "<password for your truststore>"
```

JDBC and Legacy ODBC Client Encryption Configuration

```
services.coordinator.client-endpoint.ssl.enabled: true
services.coordinator.client-endpoint.ssl.auto-certificate.enabled: false

services.coordinator.client-endpoint.ssl.keyStoreType: "type" # optional; default: JKS
services.coordinator.client-endpoint.ssl.keyStore: "path/to/keystore/jks/file"
services.coordinator.client-endpoint.ssl.keyStorePassword: "file password"
services.coordinator.client-endpoint.ssl.keyPassword: "key password"
services.coordinator.client-endpoint.ssl.trustStoreType: "type" # optional; default: JKS
services.coordinator.client-endpoint.ssl.trustStore: "path/to/truststore/jks/file"
services.coordinator.client-endpoint.ssl.trustStorePassword: "file password"
```

Intra-cluster Encryption Configuration

```
services.fabric.ssl.enabled: true
services.fabric.ssl.auto-certificate.enabled: false

services.fabric.ssl.keyStoreType: "type" # optional; default: JKS
services.fabric.ssl.keyStore: "path/to/keystore/jks/file"
services.fabric.ssl.keyStorePassword: "file password"
services.fabric.ssl.keyPassword: "key password"
services.fabric.ssl.trustStoreType: "type" # optional; default: JKS
services.fabric.ssl.trustStore: "path/to/truststore/jks/file"
services.fabric.ssl.trustStorePassword: "file password"
```

Important additional considerations

While self-signed certificates are **not recommended in any production environment, it is possible to use them** by enabling the following two properties.

```
services.fabric.ssl.enabled: true
services.fabric.ssl.auto-certificate.enabled: true
```

Starting with Dremio v24.x, all password-related entries are in the dremio.conf file can be encrypted using the dremio-admin encrypt CLI command, these entries are namely keyStorePassword, keyPassword, and trustStorePassword. On the master node, log in with the dremio user and run the dremio-admin encrypt CLI command using the following syntax:

```
./dremio-admin encrypt <string_to_encrypt>
```

Copy the entire output and use this encrypted string for the value of the password or secret in the configuration file. Restart the master node.

Authentication

Dremio supports several types of authentication, allowing it to cater to most enterprise authentication setups. The Dremio documentation covers all available options.

Dremio allows you to have both local and external authentication set up at the same time. However, please keep in mind that only one external ID provider is supported at a time and Dremio requires a unique username for each user, regardless of how the usernames are created. For example, if you have a local user whose username is user1@dremio.com, you cannot create an LDAP user whose username is also user1@dremio.com.

These are the Login Credential options available to users:

- Username / Password - User provides a username and password combination for authentication.
- Single Sign-On - User is authenticated by the configured Identity Provider, including automatic authentication, if already signed into the Identity Provider.
- Personal Access Token - User creates a private access token (PAT) for authentication, which is used in place of a username/password authentication for ODBC, JDBC, and Rest sessions.

Local (or Internal) authentication is available out-of-the-box while the remaining authentication options will need to be set up through additional configuration. Details of how to configure LDAP authentication can be found here, while for SSO - either through OpenID or AAD - the instructions can be found here or here for Okta. It is essential to remember that confidential entries such as bind passwords or client secrets can be encrypted using the dremio-admin encrypt CLI command.

Important and Additional Considerations

When using OpenID please ensure that the jwt token provided by the IdP is signed with an algorithm matching what is expected by Dremio. Discrepancy in the expected algorithm can lead to the following error `An error occurred while attempting to decode the Jwt: Signed JWT rejected: Another algorithm expected, or no matching key(s) found`

Usually, this is caused by the JWT being signed with an HMAC algorithm instead of the expected RSA. This can be confirmed by checking the jwt header and ensuring that the `alg` tag has an `RS…` algorithm rather than an `HS…` one.

The OpenID administrators should be able to provide you with an example of the jwt token being passed by their application, if not you should be able to retrieve it by sending a request to the authentication URL through a REST API testing tool such as Postman. The token will likely need to be decoded, which can be done through a decoder tool such as this. Jwt tokens are decoded and not encrypted, so please be very careful to use a trusted tool that doesn't transmit the token remotely to avoid the risk of breaches.

The decoded jwt header should look like this with an `RS…` in the `alg` tag representing RSA and `HS…` representing HMAC:

```
"header":{
      "alg":"RS256"
      "zip":"NONE"
      "typ":"JWT"
}
```

Data Security

Dremio allows you to set fine-grained permissions for users and roles on specific objects, such as datasets or functions. This is called access management, and it gives administrators the ability to control who can access what in Dremio.

Dremio provides the following access management features:

- Privileges: Privileges allow users to perform specific actions on objects in Dremio. For example, you can grant a user the privilege to view a dataset, but not to modify it. Privileges can be set on individual datasets or entire schemas.

- Row-access and column-masking policies: Row-access and column-masking policies allow you to restrict users' access to specific rows or columns in a dataset. For example, you could use a row-access policy to limit users to only seeing data that is relevant to their job role.
- Flexible management: You can configure privileges using SQL commands, REST APIs, or the Dremio user interface.
- Ownership: Dremio uses an object-oriented model to define ownership for all objects in Dremio, including sources and spaces. This means that each object has a clearly defined owner, who can transfer ownership to other users.
- Local users and roles: Administrators can create and manage users and roles in Dremio, in addition to using identities from corporate LDAP or identity providers (IdPs).

This level of control over how security controls are applied to data accessible through Dremio makes it possible to have a holistic approach to data security.

Access Management Best Practices

Identify and Categorise User Accounts

After logging in, review the current user accounts in your Dremio instance. If you're new to Dremio and plan to add user accounts, create a list of all users requiring access. Differentiate between those needing administrative or restricted access. Typically, regular users should have minimal privileges, while administrators require elevated access.

For each user or group to be added, compile a list of privileges to assign and specify the data objects they should access. When defining object access, consider Dremio's ownership delegation model and how to limit access scope. (see Ownership Delegation)

Create a Test Account

When implementing access control for the first time, consider creating a test account. This account allows you to assess the impact of various privileges and understand how users interact with data based on assigned objects. It's an excellent way to gain insights into new Dremio privileges and potential misuse scenarios.

This step is often overlooked but essential for administrators to grasp how privileges affect data access and modifications.

Upon creating a test account, apply privileges one at a time and at different object levels to observe how they restrict or enable user activities.

Establish Emergency Access Accounts

There may be situations where an administrator is unavailable or unable to access their Dremio account. In such cases, emergency access accounts become vital when users require immediate augmented access for emergency or high-priority tasks.

These accounts have elevated privileges but aren't tied to specific individuals. Trusted individuals should be provided with the login credentials for these accounts to perform necessary actions. Access to the account can be revoked by changing the password once the tasks are completed.

Implement Least Privilege

The principle of least privilege dictates that subjects should only receive the minimum necessary privileges to access an object. Granting excessive privileges can lead to unintended data alterations or acquisitions, posing security risks. This practice aligns with the "need to know" rule, where access is granted only when it's essential for the task.

When assigning privileges to users or groups, ensure they receive the minimum access required for their tasks. Avoid granting unnecessary rights. For example, if a user needs to view data on a specific object but not modify it, provide VIEW access only.

Occasionally, users or groups may require temporary additional privileges, which should be promptly revoked once the task is completed.

Communicate Access to Users

Effective communication is crucial during access control implementation. Users and groups should be informed about the specific privileges granted to them and the objects they can interact with. This ensures clarity and enables users to request additional access if necessary.

Conduct Regular Access Audits

Perform access audits at regular intervals, typically on a weekly basis. Dremio's audit logs and system tables such as sys.privileges and sys.membership, can be leveraged to set up views and dashboards to allow you to quickly see details of where access controls have been granted or changed. Review permissions granted to users and groups, considering any changes in access needs or recurring incidents.

Address issues such as accidental data deletions by reviewing the privileges responsible for such problems. Similarly, evaluate cases where additional access is frequently requested, and consider making these privileges permanent to streamline operations.

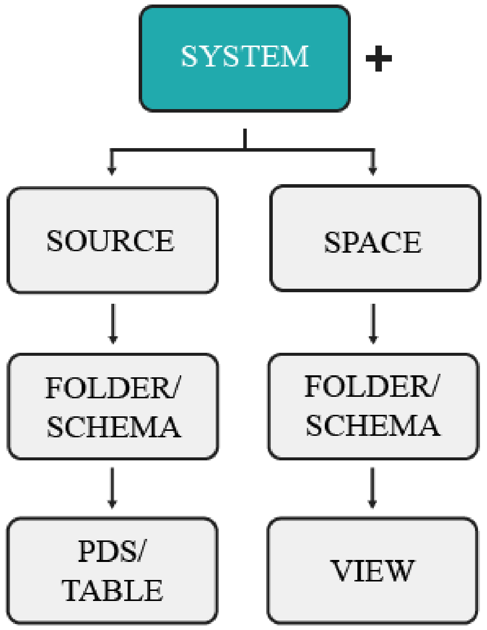
Regular access audits help resolve recurring problems and align access controls with evolving user requirements. Neglecting to periodically revisit and assess access controls can lead to inefficiencies and security risks.

Access Management Structure

Dremio employs access management, which allows for the assignment of access policies, or privileges, to users or groups.

Based on the ownership delegation model, when a user is granted privileges to an object with child objects, the user's privilege also applies to the child objects. This is also known as scope, meaning the specific objects to which a privilege applies. Access to objects is granted via privileges assigned to users or groups.

To understand how permissions are inherited between related objects, we need to review the object hierarchy within Dremio. Each object resides within a container in a hierarchy of containers. The uppermost container exists as the system user's or administrator's account. All other objects are contained within sources or spaces and organized into folders. The hierarchy of these objects is illustrated below.

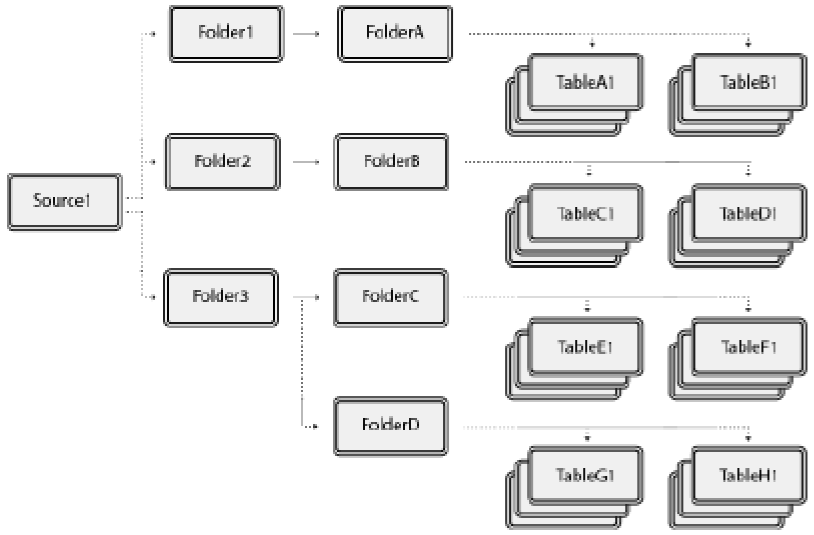


The data objects that users can access depend on how privileges are inherited. In other words, if you grant a user access to a parent object, such as a folder, they will also have access to any datasets that are already in that folder, as well as any datasets that are added to the folder in the future. For example, if you grant a user the ALL DATASETS privilege, they will only have access to existing datasets, not to the folders that contain those datasets. On the other hand, if you grant a user privileges at the source level (such as a schema), they will have access to all existing and future folders and datasets in that source.

The data object to which a user's privileges are applied is called the scope. Scopes follow a parent-child relationship. This means that you can grant users access to any object in the object hierarchy, from the top level (such as a database) down to the individual object level (such as a table).

For example, you could grant a user access to all datasets in a database, or you could grant them access to a single dataset in a specific folder.

Permissions granted to an individual table or view only give the user access to that dataset, not to the parent folder or other datasets in the same folder. So, if a user only needs access to a single dataset, you only need to grant them privileges to that object.

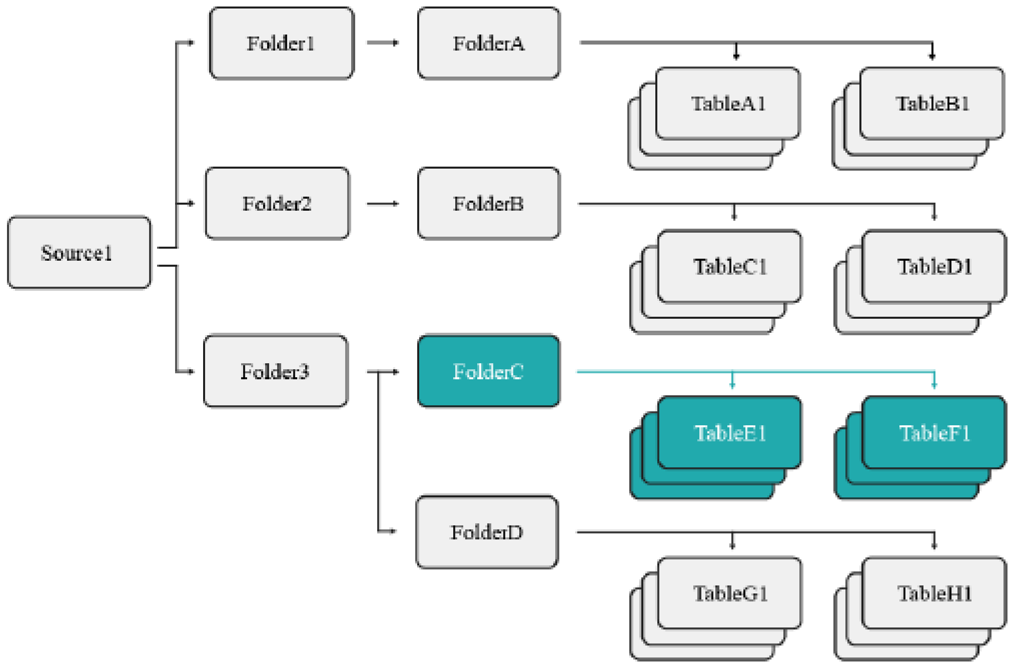


Consider the image above, which shows an example of object structure in Dremio. If a user is granted privileges to a single dataset, such as TableA1, then that is the one object they have access to. However, if a user is granted privileges at the folder level, such as Folder1, then that user's access extends to any existing and future child objects created, including FolderA, TableA1, TableB1, and so on.

Scope is a concept that describes what data objects a user or group can access. Privileges are assigned to specific data objects and determine what actions the user or group can perform on those objects. For example, you could grant a user scope to FolderA, which would give them access to all existing and future datasets in that folder and the datasets' wikis. However, they would not have access to any other folders or the source.

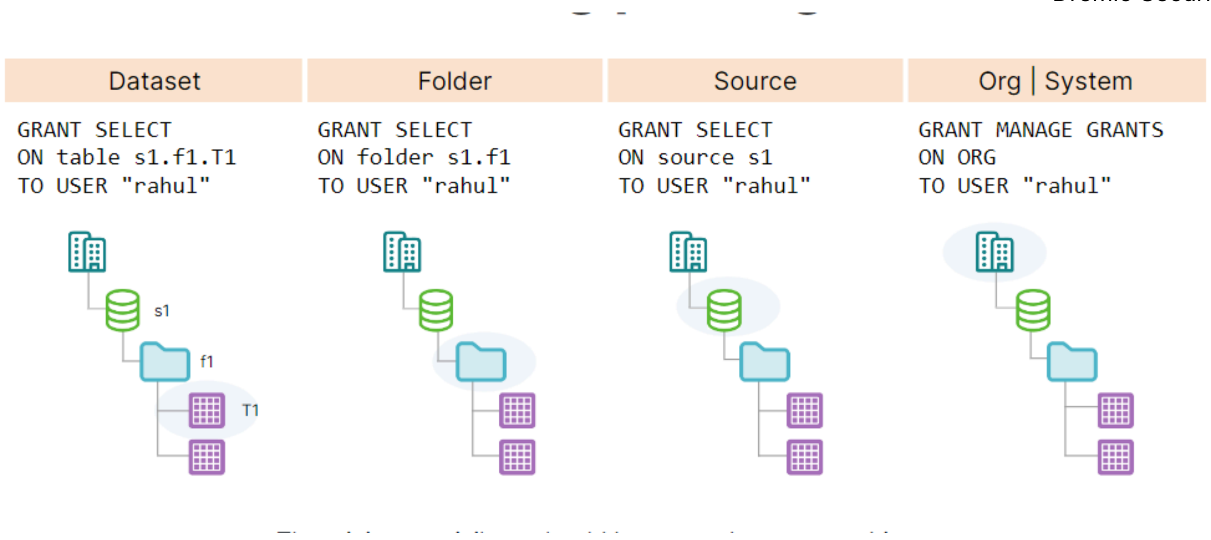
The data object that a user is granted access to depends on the inheritance model. This means it may contain child objects based on the object type. For example, if a user is granted privileges to a folder, the user's access also extends to all existing and future datasets contained in that folder.

For example, let's say that user1 is granted the SELECT privilege to the folder FolderC. This object contains multiple datasets, which the user may now access. However, a parent folder and another subfolder with its own datasets also exist.



Because of the established scope, user1 may not access FolderD because they were only granted access to FolderC's objects.

You may restrict a user's access to future and existing datasets based on the selected scope. For example, if you select a single table as the scope of a user's privilege, then that user may only perform that action to the existing dataset, as well as any future views they create using that table but, they may not access any views created from a table by another user. However, if the scope is set at the folder level, then the user may perform the granted privilege to all tables and views in that folder.



Ownership

Ownership is a security feature that controls who can access and manage an object in Dremio. Each object must have an owner, and only one owner at a time. Initially, the owner of an object is the user who created it.
The owner of an object has all privileges for that object, including the ability to:

- Grant or revoke user and role access to the object and its child objects
- Modify the object's settings
- Drop or delete the object
- To assign or modify object ownership, use the GRANT SQL command.

If an object's owner is deleted or removed, the object's access control settings may not work. Tables with restricted access can be shared with other Dremio users through the creation of views. When a user creates a view from a table they have access to, they become the owner of that new view. The user may then grant privileges to other users for that view.

This means that users can be granted access to data in a table, even if they do not have direct access to the table itself.

There are several benefits to using views to share tables:

- Views can be used to restrict access to specific columns in a table.
- Views can be used to filter the data in a table.
- Views can be used to simplify queries.
- Views can be used to improve security.

In Dremio 21.0.0 and subsequent versions, view owners can be changed only using SQL commands; editing the SQL in a view does not change the view's ownership.

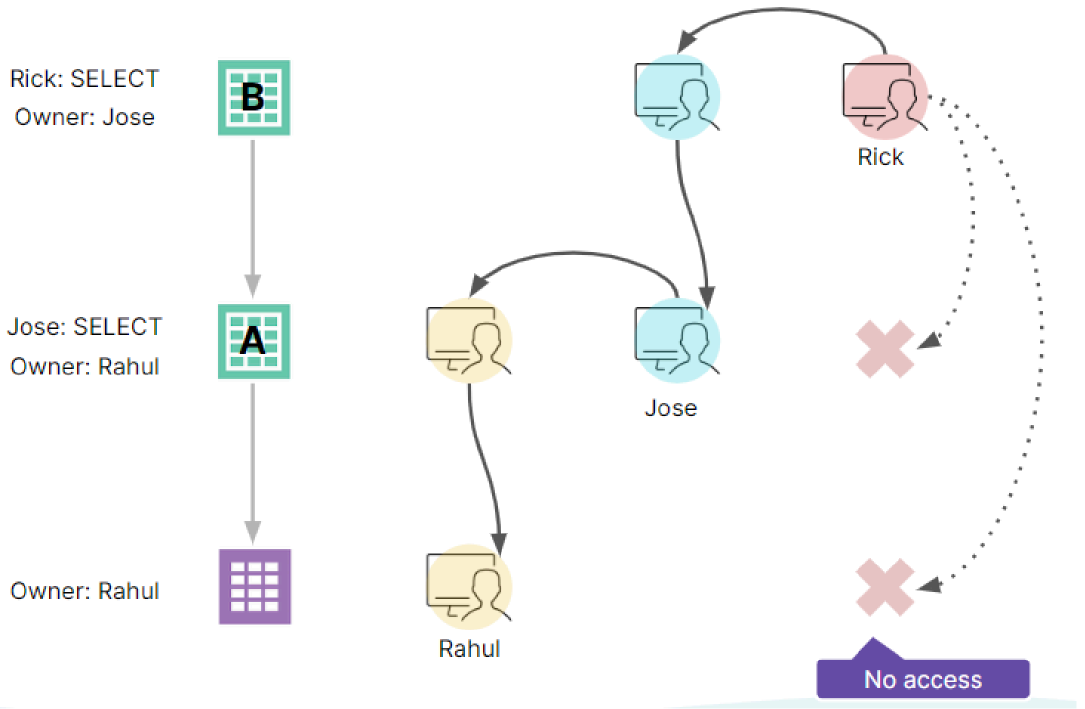
To identify dataset owners, query sys."tables" or sys.views. In the query result, the owner_id column lists the ID of the user or role that owns the dataset. The owner_type column identifies whether the owner is a user or role.

If a dataset has no owner, the owner_id value is $deleted$ in which case we will say that the dataset is orphaned. Orphaned datasets can create uncertainties in Dremio’s behavior, especially when migrating objects between environments, so we recommend running a weekly check for orphaned datasets and assigning them to an existing user.
For datasets with no owner, the administrator can use the `GRANT` SQL command to grant ownership to a user or role that has access to the table (or the underlying table, for a view).

Ownership Delegation

Ownership delegation protects sensitive data in underlying datasets while allowing users to access and interact with derived datasets. This means that an upper dataset can be shared with (and even owned by) different users without necessarily granting them access to lower datasets.

The privileges of the dataset owner are used to determine the visibility of the dataset to upper-level datasets.



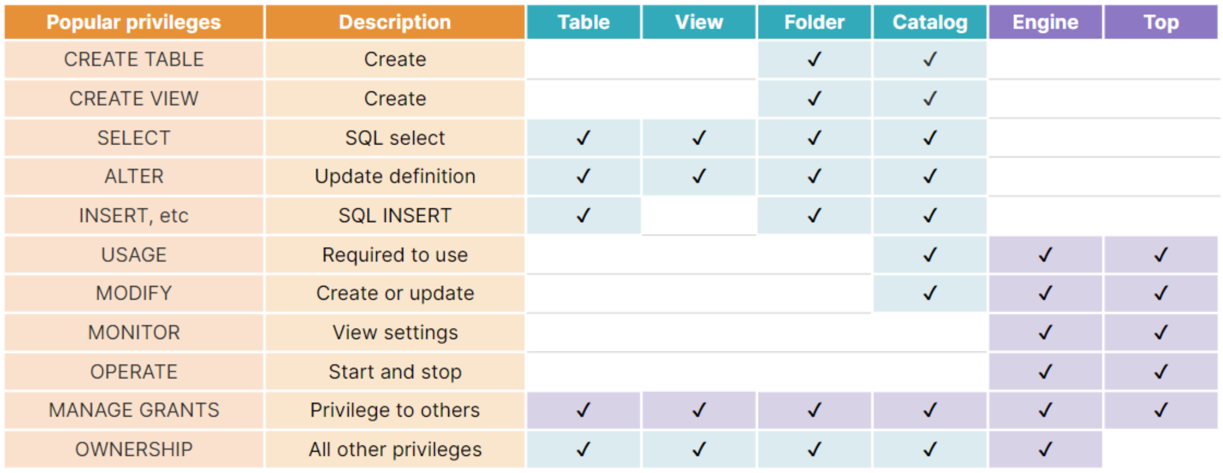
Privileges

Privileges enable users to perform explicit operations on objects in Dremio. Additionally, privileges may be set on individual datasets (tables or views) or whole schemas, allowing for a simplified configuration with larger catalogs.

By default, all users have all privileges granted to them on any objects that do not have any specific privilege grants. After a user or role is granted specific privileges on an object, access is restricted to only the users and roles who have been granted specific privileges. All other users no longer have access to the object.

Privileges can be granted using the SQL Editor, REST APIs, or the Privileges screen in the Dremio user interface (UI). The SQL Editor is accessible from any dataset, and any SQL commands that you enter apply to the scope supplied with the command itself.

There are quite a variety of privileges available within Dremio, some generic, others specific to the object that is being interacted with. The below is a handy reference for how the most popular privileges relate to the various objects.



Popular privileges	Description	Table	View	Folder	Catalog	Engine	Top
CREATE TABLE	Create			✓	✓		
CREATE VIEW	Create			✓	✓		
SELECT	SQL select	✓	✓	✓	✓		
ALTER	Update definition	✓	✓	✓	✓		
INSERT, etc	SQL INSERT	✓		✓	✓		
USAGE	Required to use				✓	✓	✓
MODIFY	Create or update				✓	✓	✓
MONITOR	View settings					✓	✓
OPERATE	Start and stop					✓	✓
MANAGE GRANTS	Privilege to others	✓	✓	✓	✓	✓	✓
OWNERSHIP	All other privileges	✓	✓	✓	✓	✓	

A detailed description and reference of all privileges divided by object can be found in the official documentation. Dataset Privileges, Sources and Spaces Privileges.

Roles and Role-Based Access Controls

In Dremio, Roles are a way to group different sets of privileges that can be assigned to users based on requirements. Rather than meticulously tracking and authorizing user permissions for specific items within Dremio, you have the option to define and assign roles tailored to various user categories within your organization that utilize Dremio. For instance, administrators often categorize roles according to company positions, such as designating a role as "Analyst." All users associated with a particular role inherit all the privileges linked to that role.

Roles can be either internal or external, depending on what kind of authentication was implemented. Internal roles are roles defined directly within Dremio, either through the UI or programmatically, facilitated by an administrator. External roles (also referred to as "groups") are those generated and overseen by an external authentication service like Okta or AAD. These groups and their respective users are not manually generated within Dremio; instead, they are automatically included when a group synchronizes with Dremio through an integrated credentials manager. Similarly, credentials for external users are managed by these services and cannot be altered via the Dremio interface.

Dremio interacts directly with the external system to retrieve and validate groups and their users as necessary. The group name stored in Dremio, visible on the Roles screen during role modification, displays the associated members as regulated by the identity manager. If a credential manager revokes a group's access to Dremio, this action does not delete the role or its associated member accounts in Dremio. These need to be removed manually.

Curating Granular Access to Data

Column Masking

Column masking represents a method for dynamically obfuscating or altering private data at the column level just before executing a query. As an illustration, a table or view's owner can implement a policy on a column to reveal only the year portion of a date or the last four digits of a credit card number.

Column masking policies can take the form of any User-Defined Function (UDF) with a scalar return type that matches the data type of the column to which it's applied. However, it's crucial to note that each column can have only one masking policy applied.

Row Access

Row access policies offer a means of regulating which records within a table or view are accessible to specific users and roles. For instance, a table or view owner might enforce a policy that excludes customers from a particular country unless the querying user possesses a specific role.

Row access policies can take the form of any boolean UDF applied to the table or view. The UDF's return value is logically treated in a query as if it were an AND operator included within a WHERE clause. It's essential to ensure that the return type of the UDF is BOOLEAN; otherwise, Dremio will raise an error during execution.

UDFs for Filtering and Masking Policies

A UDF is a callable routine defined by a user that accepts input parameters, carries out its designated function, and returns either a scalar value or a set of rows.

The UDFs that underpin filtering and masking policies must be defined independently of the data sources. This approach serves a dual purpose: it enables organizations to apply a single policy across multiple tables and views. It also improves security by limiting access to policies and deterring unauthorized alterations. Any adjustments made to a single UDF automatically propagate to policies associated with tables or views employing that particular access or mask policy.

The policies in Dremio are enforced as follows:

1. An administrator user with the ADMIN role builds a UDF to function as a security policy.
2. The administrator associates the security policy with one or more tables, views, and/or columns.
3. Dremio enforces the policy at runtime when an end-user executes a query.

The creation of UDFs and the attachment of security policies are accomplished through SQL commands. These policies come into play prior to execution, during the query planning phase. During planning, Dremio conducts a preliminary scan of the table or view for row-access policies and subsequently evaluates each accessed column for column-masking policies. If any policies are present, they are automatically applied to the relevant scope using the associated UDF within the query plan.

UDFs built for row-access or column-masking purposes operate as an "implicit view," replacing a table or view reference within an SQL statement before the query is processed. This implicit view's creation hinges on examining policies applied to a table, view, or column.

Setting a Policy

To create a row-access or column-masking policy, you must perform the following steps using the associated SQL commands:

1. Create a new UDF or replace an existing one using the `CREATE [OR REPLACE] FUNCTION` command.
2. Create a policy to apply the function using either `ADD ROW ACCESS POLICY` for row-level access or `SET MASKING POLICY` for column-masking. These may be used either when creating the table/view or after creating it, invoking an `ALTER` command.

It is necessary for the function to accept at least one input with a data type consistent with the column that will be used to apply the policy. Functions that don't require input are not good candidates for a security policy.

Auditing

For organizations subject to compliance and regulation where auditing is regularly required, Dremio offers full audit logging, wherein all user activities performed within Dremio are tracked and traceable via the audit.json file. Each time a user performs an action within Dremio, such as logging in or creating a view, the audit log captures the user's ID and username, object(s) affected, action performed, event type, SQL statements used, and more.

Audit logging is enabled by default and is available only to users with administrative rights at the System level. The log file location may be configured via the dremio.log.path property in the dremio-env file, where location, file size, and rotation schedule can also be defined.

As well as logging logins, the audit.json file will contain a trace of creations, updates, and deletes across all Dremio objects. The full list of tracked events can be found here.

As well as the audit.log file, especially in situations where compliance rules are very stringent, or for datasets containing PII, the queries.json file will contain a list of all executed jobs for the past 30 days (configurable). This can be a valuable resource for monitoring access to specific datasets and requests. If scraping and monitoring of logs can be achieved, it would be good practice to have an event listener trigger alerts for queries that access specific datasets, and columns or apply filters on identifying data.