Role-based data management best practices for a Splunk deployment

Generated: 5/20/2019 3:34 pm
Role-based data management best practices for a Splunk deployment

The Splunk roles feature enables you to define permissions and capabilities for a collection of users, for example, setting search limitations, providing access to product features, data, and knowledge objects, and setting the default app users land in when they log into Splunk. When data becomes more diverse and the number of users increases, it can be more complex to manage roles. This topic explores how you can manage roles, access to data, and access to product features in a more modular and scalable way.

Role separation is a process that segregates a user's access to data from their access to Splunk capabilities. Data governance is a process that controls access to certain data and capabilities for certain roles. Introducing a naming convention and separating roles and features can enable you to implement a highly flexible and scalable role-based access control (RBAC) solution.

This article does not discuss role inheritance. That is deliberate so you can clearly see direct roles and access permissions without having to consider layers of permissions inherited from other roles.

Audience

- Architect
- Engineer
- Search expert
- Knowledge manager

For more about these roles, see Roles best practices.

Guidelines for implementing role control and data management

To make role control more granular, you can organize user access requirements into functional categories, such as:

- Data access
- Search restrictions
- Product feature capabilities
- Knowledge object permissions
- Default app
For example, a role classified as data access would define access to data, but not search restrictions, product feature capabilities, or knowledge object permissions. Here’s more about each of these five categories to better understand why such segregation can be useful.

**Data access (data)**

The `srchIndexesAllowed` parameter of `authorize.conf` is where you can specify which indexes a user is allowed to search. See Add and edit roles with `authorize.conf` in the *Securing Splunk Enterprise* for more information.

Rather than defining a role for data access per index, you can create collections based on commonality, like function. For example, `data_os` where `srchIndexesAllowed = windows, os, linux`.

An index naming convention enables you to use wildcards, which becomes a boon to your solutions scalability. For details about how to establish a good naming convention, see In depth: Naming conventions.

**Search restrictions (search)**

Search restrictions enable you to develop a more flexible maturity model for managing user access. You can provide new users a role like `search_new` that restricts access to certain search functions (attributes of `authorize.conf` with the string `srch`) so that new users can’t make lasting mistakes on the platform as they learn.

For more information about authorization for product capabilities, search capabilities, and data, see About defining roles with capabilities in *Securing Splunk Enterprise*.

**Product feature capabilities (feature)**

A best practice with product feature capabilities is to limit access to features that can impact platform performance, such as scheduled searches and report accelerations, to those who have the technical training and architectural endorsement to use them. You can restrict access to these types of features for new users to limit risk to the system while they learn.

For more information about authorization for product capabilities, search capabilities, and data, see About defining roles with capabilities in *Securing Splunk Enterprise*.

**Knowledge object permissions (ko)**
As teams collaborate and create insights into their data, they will want to share knowledge objects with other users. A best practice to improve search performance is to establish more granular control over which groups can globally read knowledge objects. For example, if a security team uses many security-related tags and fields for their data models, they can limit the search parsing impact of sharing content with the web team by removing read access to the knowledge objects the web team doesn’t need.

The definition of the app and global knowledge object permissions is within the `default.meta` and `local.meta` from the app where the knowledge object was defined. For more about `Defaultmeta.conf`, see `Defaultmeta.conf` in the *Splunk Enterprise Admin Manual*.

For information on knowledge object permissions, see Manage knowledge object permissions in the *Knowledge Manager Manual*.

Default app (workspace)
When you create an app as a workspace, users enjoy a more comfortable and straightforward experience when they log into Splunk and land directly in an app tailored to their team or use case. They avoid having to hunt for the right place to start, and they can easily find the knowledge objects they need to do their jobs. See Workspace best practices.

To make an app a default workspace landing page, add the app name to the `default_namespace` attribute in `user-prefs.conf`.

For information on configuring a default app, see Configure Splunk Web to open directly to an app in the *Splunk Enterprise Admin Manual*.

**Guidelines for using naming conventions to make access more modular**

While Splunk's role inheritance helps manage the sprawl of roles, larger environments can struggle to keep connections and lines of access straight if there are layers of inheritance to sort. To address this, you can create roles that manage access to a particular capability without affecting other capabilities. To simplify this modular approach, you can use a naming convention for the roles you create like this: `<capability>_<descriptor>`.

Here are how the different `<capability>` of a role can be translated for this naming convention:

- `data` for data access
• search for search constraints
• feature for product feature capabilities
• ko for knowledge object permissions
• workspace for default app

A <descriptor> describes the purpose of that role. Putting it together with the <capability> you have examples like:

• data_operatingsystems for a role that defines operating system-related indexes allowed for searching
• workspace.middleware for a role that defines the default app the middleware team will see when they log in
• search.new for a role for new Splunk users that restricts access to limit system impact
• feature.architecture for a role that unlocks distributed deployment capabilities
• ko.marketing for a role that is used with Splunk permissions to provision read/write access to knowledge objects

Provisioning a group's role

Once you establish modular groups like the example above, you can assign any given team within Splunk roles that are an aggregate of these different modular groups. Groups can be constructed from at least one selection of data, search, feature, ko, and workspace components.

Use this approach only if your Splunk implementation has scaled to the point that such differentiation makes sense. If your Splunk implementation is fairly simple, your needs may be addressed sufficiently with the traditional approach to role management.

Promoting users

Over time, business changes may require that a given user’s role evolve. This model supports such changes because users can just be made members of new groups to give them the new access they need.

For example, a user could be a member of the data_operatingsystems group, but now also needs access to web data. So you can just add them to the data_web group. In another example, a user is being promoted to perform more advanced search operations. So while the user is a member of search_user, you would also add the user to search_alerting, which provides the user the search capabilities to create and schedule searches that have alert functionality.
In both these scenarios, the user’s capabilities were changed without needing to change the underlying Splunk configuration of these roles. That means you can implement this business need without a change approval or platform restart.

**Manage access to data and capabilities using Active Directory groups**

Membership to each of the groups may be implemented through third party solutions and connected to Splunk with SAML, SSO, and/or LDAP. You can map those user group definitions directly to Splunk roles.

Here is an example of some of the roles available within a hypothetical customer’s Active Directory platform. In this situation, users are made members of at least one of each type of group.

![Group Membership Example](image)

**Before working in conf files**

The configuration file `authorize.conf` contains about 100 parameters you can use to fine tune permissions broadly, such as by role `[role_<roleName>]`, or narrowly, such as by list settings `[capability::list_settings]`. Splunk Web exposes some, but not all of these settings.

This article refers to making edits to `authorize.conf`. Before working with configuration files, see these topics in the *Admin Manual* to get familiar with the nuances of configuration file structure, configuration file directories, configuration file precedence, and when to restart Splunk Enterprise after a configuration file change for a direct `.conf` edit to apply. To help keep it all straight, Splunk provides `btool`, a command-line utility, to troubleshoot issues with `.conf` file interactions and precedence.