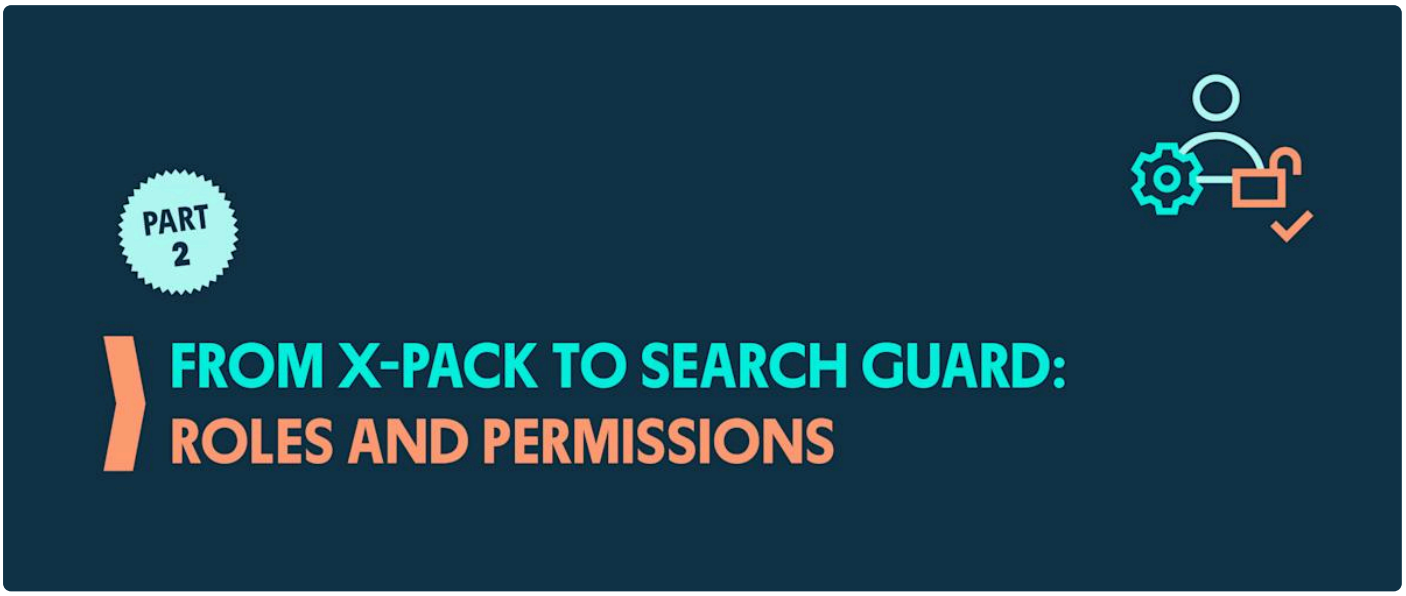
# From X-Pack to Search Guard: Part 2 - Roles and Permissions

Jochen Kressin · 2025-06-23




This article demonstrates how to migrate custom roles from X-Pack Security to Search Guard, enabling you to maintain your access control structure during the transition between these security solutions.

## X-Pack Roles

In X-Pack, privileges are pre-defined sets of actions for operating on indices or the Elasticsearch cluster. For example, `write` is an index-level privilege that grants write operations on documents, while the cluster-level privilege `create_snapshot` allows creating snapshots and listing existing repositories. Privileges represent the lowest granularity in X-Pack Security. You cannot modify pre-defined privileges or work directly at the action level.

Permissions are created by combining one or more privileges with a secured resource (typically an index or index pattern). For example, a permission might grant `read` and `write` privileges to all indices matching the `my-index*` pattern.

A role, which can be assigned to users through role mapping, is a named set of permissions:

© 2026 floragunn GmbH · search-guard.com

```
{
  "cluster":[
    "all"
  ],
  "indices":[
    {
      "names":[
        "index*",
        "some"
      ],
      "privileges":[
        "all"
      ],
      "allow_restricted_indices":false
    },
    {
      "names":[
        "index1*",
        "some1"
      ],
      "privileges":[
        "all"
      ],
      "allow_restricted_indices":false
    },
    {
      "names":[
        "index2*",
        "some1"
      ],
      "privileges":[
        "read"
      ],
      "allow_restricted_indices":false
    }
  ]
}
```

Roles and role mappings can be retrieved through the X-Pack Security APIs or by querying the `.security` index directly.

## Search Guard Roles

In Search Guard, similar concepts exist, but with key differences in terminology and capabilities. Unlike X-Pack Security, Search Guard allows you to define permissions at the action level. What X-Pack calls "privileges" are "action groups" in Search Guard:

| X-Pack Security | Search Guard | Remarks |
|---|---|---|
| action | action | Directly configurable in Search Guard |
| privilege | action group | |
| permission | permission | |

| X-Pack Security | Search Guard | Remarks |
|---|---|---|
| role | role | |
| role mapping | roles mapping | |

A role in Search Guard looks like this:

```
{
  "cluster_permissions":[
    "SGS_CLUSTER_ALL"
  ],
  "index_permissions":[
    {
      "index_patterns":[
        "index*",
        "some"
      ],
      "allowed_actions":[
        "SGS_INDICES_ALL"
      ]
    },
    {
      "index_patterns":[
        "index1*",
        "some1"
      ],
      "allowed_actions":[
        "SGS_INDICES_ALL"
      ]
    },
    {
      "index_patterns":[
        "index2*",
        "some1"
      ],
      "allowed_actions":[
        "SGS_READ"
      ]
    }
  ]
}
```

## Migrating Roles

To migrate custom roles, we need to retrieve them from Elasticsearch, transform their structure to match Search Guard's format, map X-Pack privileges to Search Guard action groups, and import them into Search Guard. We'll demonstrate a practical approach using command-line tools.

For processing JSON on the command line, we recommend installing jq and using the `bash` shell for the commands below.

First, retrieve all your custom roles from Elasticsearch:

```
curl -Ss -u "elastic:<password>" localhost:9200/.security/_search?pretty\&size=1000\&q=type:role
```

Use `jq` to transform the roles into a format closer to Search Guard's requirements:

```
curl -Ss -u "elastic:<password>" localhost:9200/.security/_search?pretty\&size=1000\&q=type:role |
jq '.hits.hits[] | { _rolename: ._id | split("-")[1], cluster_permissions: ._source.cluster,
index_permissions: ._source.indices } | del(.index_permissions[].allow_restricted_indices)'
```

This produces output like:

```
{
  "_rolename": "my_custom_role",
  "cluster_permissions": [
    "all"
  ],
  "index_permissions": [
    {
      "names": [
        "index*",
        "some"
      ],
      "privileges": [
        "all"
      ]
    },
    {
      "names": [
        "index1*",
        "some1"
      ],
      "privileges": [
        "all"
      ]
    },
    {
      "names": [
        "index2*",
        "some1"
      ],
      "privileges": [
        "read"
      ]
    }
  ]
}
```

Now, we need to rename fields and map privilege names to Search Guard action groups. Let's enhance our approach with a little bash script:

© 2026 floragunn GmbH · search-guard.com

```
#bin/bash

# Initialize an empty JSON object
echo "{}" > sg_roles.json

curl -k -Ss -u "elastic:4JidJvyIIcbo2INER2Wh" https://localhost:9200/.security/_search?
pretty\&size=1000\&q=type:role | jq '[.hits.hits[] | { _rolename: ._id | split("-")[1],
cluster_permissions: ._source.cluster, index_permissions: ._source.indices } |
del(.index_permissions[].allow_restricted_indices) |
del(.index_permissions[].allow_restricted_indices)]' > roles_mig_tmp.json

# Loop roles_mig.tmp (every row is a role)
for row in $(cat "roles_mig_tmp.json" | jq -r '.[] | @base64'); do

   # Count index permissions in this role
   index_permissions_count=$(echo ${row} | base64 --decode | jq  '.index_permissions | length')
   let index_permissions_count=index_permissions_count-1

   JQ_NAME_FIX=""
   for i in $(eval echo "{0..$index_permissions_count}")
   do
     # Rename .names to .index_patterns and .privileges to .allowed_actions
     JQ_NAME_FIX=".index_permissions[$i].index_patterns=.index_permissions[$i].names |
$JQ_NAME_FIX "
     JQ_NAME_FIX=".index_permissions[$i].allowed_actions=.index_permissions[$i].privileges |
$JQ_NAME_FIX "
   done

   # Extract the role name
   ROLE_NAME=$(echo ${row} | base64 --decode | jq  '._rolename' | tr -d '"')

   # Remove fields we no longer need
   JSON="$(echo ${row} | base64 --decode | jq  "$JQ_NAME_FIX del(.index_permissions[].names) |
del(.index_permissions[].privileges) | del(._rolename)")"

  # Append the role to the final JSON object
  tmp=$(mktemp)
  jq --arg key "$ROLE_NAME" --argjson val "$JSON" '. + {($key): $val}' sg_roles.json > "$tmp" &&
mv "$tmp" sg_roles.json

done
```

## Mapping Privileges to Action Groups

Next, we need to replace the `privileges` from the original X-Pack role definition with Search Guard `Action Groups`. Open the `sg_roles.json` file that was generated above and search and replace the privileges in the file.

The following tables map common X-Pack privileges to Search Guard action groups. Although some names may sound similar, the granted permissions may differ slightly. Review each privilege to ensure the corresponding action group doesn't grant excessive permissions.

If no default action group meets your needs, you can combine multiple action groups or define custom action groups.

© 2026 floragunn GmbH · search-guard.com

### Cluster Privileges

| X-Pack Security privilege | Search Guard action group |
|---|---|
| all | SGS_CLUSTER_ALL |
| create_snapshot | SGS_MANAGE_SNAPSHOTS |
| manage*index*templates | SGS_CLUSTER_MANAGE_INDEX_TEMPLATES |
| manage*ingest*pipelines | SGS_CLUSTER_MANAGE_PIPELINES |
| monitor | SGS_CLUSTER_MONITOR |

### Indices Privileges

| X-Pack Security privilege | Search Guard action group |
|---|---|
| all | SGS_INDICES_ALL |
| create | SGS_CREATE_INDEX |
| create_index | SGS_CREATE_INDEX |
| delete | SGS_DELETE |
| index | SGS_WRITE |
| manage | SGS_MANAGE |
| monitor | SGS_INDICES_MONITOR |
| read | SGS_READ |
| write | SGS_WRITE |

## Using sgctl to connect to our Search Guard secured cluster

In the following steps, we will use the Search Guard Control Tool (sgctl) to connect to the cluster and upload the user configuration. First, download the latest version of sgctl from our Maven repository:

https://maven.search-guard.com/search-guard-flx-release/com/floragunn/sgctl/

Download the ZIP file und unzip it in a directory of your choice. Next, we connect to Search Guard by specifying the Root CA and the certificate and key of our admin certificate. If you have used the Search Guard Demo Installer, the command looks like:

```
./sgctl.sh connect localhost --ca-cert /path/to/elasitcsearch/config/root-ca.pem --cert
/path/to/elasitcsearch/config/kirk.pem --key /path/to/elasitcsearch/config/kirk-key.pem
```

If your certificates are named differently or stored elsewhere, update the path settings accordingly. If successful, the output looks like:

© 2026 floragunn GmbH · search-guard.com

```
Successfully connected to cluster mycluster (localhost) as user
CN=kirk,OU=client,O=client,L=test,C=de
```

## Upload the roles configuration

The last step is to simply upload the roles definition to Search Guard:

```
./sgctl.sh update-config /path/to/sg_roles.json
```

Which should create an output like this:

```
Successfully connected to cluster mycluster (localhost) as user
CN=kirk,OU=client,O=client,L=test,C=de
Configuration has been updated
```

## Articles in this series

- From X-Pack to Search Guard: Part 1 - TLS Setup
- From X-Pack to Search Guard: Part 2 - Roles and Permissions (This article).

Coming Soon:

- From X-Pack to Search Guard: Part 3 - Users
- From X-Pack to Search Guard: Part 4 - Document- and Field-Level Security

---

This article was published on the Search Guard blog: https://search-guard.com/blog/from-x-pack-to-search-guard-part-2-roles-and-permissions/
Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · Start a free trial