What's new in Search Guard 6: Attribute based document access

Claudia Kressin · 2018-01-15

Search Guard 6 is out for a couple of weeks now, and it contains many new and compelling security features. In this series, we will look at the main new features and show how they can help you improving your security infrastructure and staying compliant with regulations like GDPR, HIPAA or PCI. I will start with one of my personal favourites: Fully dynamic document access control based on user attributes.

Search Guard always had document- and field-level security built in. With document-level security (DLS) you can control access to documents in an index based on an Elasticsearch query. With field-level security (FLS) you can define which fields are visible for a Search Guard role and which are not.

Let's say we have an employee index which contains personal records of all employees of a company, like:

```
{
   "FirstName":"ELLIOT",
   "LastName":"CASTRO",
   "Designation":"CEO",
   "Salary":250000,
   "Address":"327 West Orchard RoadMiami Gardens, FL 33056",
   "Gender":"Male",
   "Age":61,
   "MaritalStatus":"Married",
   "Interests":"Reading,Playing music,Aircraft Spotting"
}
```

You can restrict access to this sensitive data by defining a DLS query that filters out certain documents and fields. Say we have a role called human_resources_trainee. We don't want this role to be able to see the record of the CEO, and we don't want to show the *Salary*, *Address* and *Gender* field. This can be done quite easily by defining a DLS query and FLS field exclusion list like:

```
human_resources_trainee:
  cluster:
    - CLUSTER_COMPOSITE_OPS_RO
  indices:
    'humanresources':
      '*':
        - READ
      _dls_: '{ "bool": { "must_not": { "match": { "Designation": "CEO" }}}}'
      _fls_:
        - '~Salary'
        - '~Address'
        - '~Gender'
```

You can use the full range of Elastic's query DSL here, so the query can be as simple or complex as needed.

Making the query dynamic

This is an entirely powerful feature already, but you probably noticed there is one big limitation: The query is static. What if you want to show users only their documents? Or only records that have the same *Designation* as the current user? With static DLS queries, this is cumbersome – you would need to set up one role for each *Designation* in your company, and then carefully map the users to their correct role. Not practical.

What we really want are dynamic DLS queries, where you can use placeholder variables that get substituted by attributes of the user at runtime.

Adding the username to the query

Search Guard always had the feature to add the username of the currently logged in user as variable to the DLS query. Say you have a field called *owner* in your documents, and this field contains the username, you can write a DLS query like:

```
_dls_: '{ "bool": { "must": { "match": { "owner": "${user.name}" }}}}'
```

Search Guard will now substitute the variable `${user.name}` with the username of the currently logged in user.

Unleashing the full power of dynamic DLS queries

With Search Guard 6, we took variable substitution to another level, making it possible to configure fully dynamic access to documents in your index, based on a user's attributes. This makes it possible to build ultra-flexible access control on document level, without any configuration overhead.

Using custom attributes from Active Directory, LDAP, and JWT in DLS queries

In an enterprise environment, you usually authenticate your users against centralized identity providers like Active Directory, LDAP or IDPs that issue JSON web tokens. Apart from the user's credentials, these systems also store additional information and data, for example, employee number, title, department and security-related data. Why not make this data available in any DLS query, and use it to make it fully dynamic? That's exactly what dynamic DLS queries, already released with the first beta of Search Guard 6, are about. You can now use:

- any attribute stored in an Active Directory / LDAP entry
- any claim in a JSON web token

Active Directory / LDAP Example

Let's say the user Elliot Castro has the following attributes stored in Active Directory / LDAP:

Attribute Description	Value
objectClass	***inetOrgPerson (structural)***
cn	**Elliot Castro**
sn	**CASTRO**
businessCategory	CEO
givenName	ELLIOT
mail	elliotcastro@example.com
uid	elliotcastro
userPassword	Plain text password

Search Guard makes all of these attributes available in any DLS query as variables. Simply prefix the attribute with "*attr.ldap*" and use it like the *username* attribute in the example above.

In our example, the standard LDAP attributes *sn (surname)* and *givenName* map to the *FirstName* and *LastName* fields of the employee records. We can now use these attributes to limit access to only the user's own record like:

```
sg_own_record:
  cluster:
    - CLUSTER_COMPOSITE_OPS_RO
  indices:
    'humanresources':
      '*':
        - READ
      _dls_: '{ "bool": { "must": [{ "match": { "FirstName": "${attr.ldap.givenName}" } }, {
"match": { "LastName": "${attr.ldap.sn}" } } ] } }'
```

The user will only be able to see records where the *FirstName* field matches the *givenName* field of the LDAP entry, and where the *LastName* field matches the *givenName* field. In other words: Only his / her own record.

JWT example

You can also use claims in a JSON web token as part of the dynamic query. A JWT for our sample user *elliotcastro* might look like:

```
{
  "uid": "elliotcastro",
  "mail": "elliotcastro@example.com",
  "sn": "ELLIOT",
  "givenName": CASTRO,
  "Designation": "CEO",
  ...
}
```

The DLS query we used above looks exactly the same for JWT, the only difference is that instead of the "*attr.ldap*" you use *"attr.jwt"* to access the attributes:

```
sg_own_record:
  cluster:
    - CLUSTER_COMPOSITE_OPS_RO
  indices:
    'humanresources':
      '*':
        - READ
      _dls_: '{ "bool": { "must": [{ "match": { "FirstName": "${attr.jwt.givenName}" } }, {
"match": { "LastName": "${attr.jwt.sn}" } } ] } }'
```

Towards GDPR compliance with dynamic queries

GDPR is coming, and it affects how companies can use and process *Personally Identifiable Information (PII)*. Art. 6 regulates the *Lawfulness of processing* PII data and states:

> Processing shall be lawful only if and to the extent that at least one of the following applies:

> 1. the data subject has given consent to the processing of his or her personal data for one or more specific purposes;

For example, a customer might have agreed to get newsletters and personalized ads, but does not give consent to process his or her PII for statistical and marketing purposes. Companies need to obey this regulation and use PII only for purposes the customer has explicitly given consent to. You will likely want to store the given consent in your customer records, e.g.:

```
{
   "FirstName":"Jane",
   "LastName":"Roe",
   "CustumerNumber":"123",
   "GDPR_Purpose":["newsletter", "ads"],
   ...
}
{
   "FirstName":"John",
   "LastName":"Doe",
   "CustumerNumber":"456",
   "GDPR_Purpose":["marketing", "newsletter", "statistics"],
}
```

Now imagine that your employee records contain a field *visible*gdpr*purpose* which controls which PII this employee is allowed to see and process. In a JWT, this might look like:

```
{ "uid": "elliotcastro",
  "mail": "elliotcastro@example.com",
  "visible_gdpr_purpose": "marketing statistics"
}
```

Instead of setting up different roles for each purpose and mapping users to these roles, you can simply use one role with a dynamic DLS query. This query will only show those records to the logged in user where the at least one *visible*gdpr*purpose*term matches the *GDPR*Purpose_ terms of the customer record:

```
sg_own_record:
  cluster:
    - CLUSTER_COMPOSITE_OPS_RO
  indices:
    'humanresources':
      '*':
        - READ
      _dls_: '{ "bool": { "must": { "match": {"GDPR_Purpose":
"${attr.jwt.visible_gdpr_purpose}"}}}}'
```

Summary

Search Guard 6 introduces fully dynamic DLS queries which make it possible to use additional user attributes from Active Directory, LDAP or JSON web tokens as part of the query. With this attribute-based access control to documents, you can build powerful, flexible and dynamic permission schemes

with minimum configuration overhead. This new feature will also help you to meet compliance regulations like HIPAA, GDPR, PCI or SOX.
